MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

Mariana Miloșescu





Informatică

Profilul real

Manual pentru clasa a IX-a

Specializarea: matematică, informatică, științe ale naturii

EDITURA DIDACTICĂ ȘI PEDAGOGICĂ, R.A.
2006



MINISTERUL EDUCAŢIEI ŞI CERCETĂRII

Mariana Miloşescu

Informatică

Profilul real

Specializarea: matematică-informatică, ştiinţe ale naturii

Manual pentru clasa a IX - a



EDITURA DIDACTICĂ ŞI PEDAGOGICĂ, R.A.
BUCUREŞTI, 2006

1. Informatica şi societatea

1.1. Prelucrarea informaţiilor

Calculatorul a fost inventat de om pentru a prelucra informaţia. El îl ajută să prelucreze *foarte uşor*, într-un timp *extrem de scurt*, cu *foarte mare acurateţe*, o *mare cantitate* de informaţie *foarte complexă*.

Prelucrarea informaţiei este veche de când lumea. **Prelucrarea voluntară a informaţiei** s-a făcut însă abia atunci când babilonienii au scris primele semne cuneiforme pe tăbliţele de lut. Aşadar, prima manifestare a prelucrării informaţiei a fost *scrisul*. Încă din antichitate, pot fi puse în evidenţă două tipuri de prelucrări de informaţii:

- ✓ **prelucrarea textelor = scrisul;**
- ✓ **prelucrarea numerelor = calculul numeric.**

Prelucrarea automată a informaţiei a fost posibilă o dată cu apariţia calculatoarelor electronice. Aşadar, scopul utilizării unui calculator este de a prelucra **informaţia**. Informaţia prelucrată poate fi formată din texte, numere, imagini sau sunete şi este păstrată pe diferite medii de memorare, în diferite formate, sub formă de **date**.

Transformarea datelor în informaţii nu este un atribut exclusiv al calculatorului. Acest fenomen a apărut o dată cu omul. De la primele reprezentări ale unor cantităţi cu ajutorul degetelor, al pietricelelor sau al beţişoarelor, şi de la manipularea manuală a acestor obiecte pentru a afla câte zile mai sunt până la un anumit eveniment sau câte animale au fost vânate sau câţi războinici are tribul vecin, putem spune că are loc un proces de transformare a datelor în informaţii. Degetele, pietricelele şi beţele reprezintă datele, iar ceea ce se obţine prin manipularea lor (numărul de animale vânate, numărul de zile, numărul de războinici) reprezintă informaţia. Cele ce deosebesc un astfel de proces de o prelucrare cu ajutorul calculatorului sunt viteza de obţinere a informaţiilor şi modul de reprezentare a informaţiilor sub formă de date.

Calculatorul nu ştie să prelucreze decât şiruri de cifre binare, care pot fi modelate fizic prin impulsuri de curent, cu două niveluri de tensiune, ce corespund celor două cifre binare: 0 şi 1. Prin urmare, datele vor fi codificări binare ale informaţiei existente în exteriorul calculatorului. Dacă, într-o prelucrare manuală, datele sunt reprezentate de obiecte care pot fi manipulate de om (beţişoare, pietricele sau degete), în cazul unei prelucrări automate datele vor fi reprezentate prin obiecte pe care le poate manipula calculatorul, adică şiruri de biţi.

Aşadar, din punct de vedere al unei prelucrări automate a informaţiei, diferenţa dintre dată şi informaţie este:

- ✓ **Informaţia** este un mesaj care înlătură necunoaşterea unui anumit eveniment şi are caracter de noutate. Informaţiile sunt interpretate de oameni.
- ✓ **Data** este reprezentarea informaţiei în interiorul calculatorului. Calculatorul nu înţelege conţinutul acestor date, el numai le prelucrează, prin operaţii specifice fiecărui tip de dată. În urma prelucrării datelor, calculatorul poate furniza omului informaţii.

Pentru a rezolva o anumită sarcină, trebuie să cunoaştem modul în care se poate face acest lucru. Aşadar, trebuie să găsim o anumită **metodă**, adică un set de paşi pe care trebuie să-i executăm ca să realizăm sarcina. Acest set de paşi formează **algoritmul** pentru rezolvarea problemei respective. Inițial, studiul algoritmilor a fost o disciplină a matematicii, prin care se căuta să se găsească un set unic de instrucțiuni prin care să se descrie rezolvarea oricărei probleme dintr-o anumită categorie. Ați învățat deja o parte din aceşti algoritmi: algoritmul lui Euclid pentru găsirea celui mai mare divizor comun dintre două numere, algoritmul împărțirii unui număr, algoritmul extragerii rădăcinii pătrate dintr-un număr, algoritmul conversiei unui număr reprezentat în baza zece într-un număr reprezentat într-o altă bază de numerație etc. Cu timpul, descrierea metodei de rezolvare a unei probleme cu ajutorul algoritmilor s-a extins şi în alte domenii de activitate.

La fel ca şi omul, şi un calculator, pentru a putea rezolva o anumită sarcină, trebuie „să aibă cunoştințe" despre modul în care se poate face aceasta, adică să cunoască algoritmul de rezolvare a problemei. Această informație i se transmite calculatorului prin intermediul unui **program**. Deoarece **limbajul natural** (limbajul prin care comunică oamenii) nu este înțeles de calculator, care este construit astfel încât să poată prelucra numai cifre binare, programul prin care i se comunică algoritmul este scris într-un limbaj de programare. **Limbajul de programare** este un **limbaj artificial** care, prin exprimări simbolice (**instrucțiuni**), descrie operațiile de prelucrare pe care trebuie să le execute calculatorul. El îi permite omului să comunice cu calculatorul (să îi dea comenzi pe care să le execute), deoarece fiecare instrucțiune din limbajul de programare va fi tradusă într-un grup de **instrucțiuni maşină** (instrucțiuni în **limbaj maşină**), adică un şir de biți care au o anumită semnificație pentru calculator. Acest limbaj se numeşte limbaj maşină deoarece este propriu fiecărui tip de maşină (calculator), fiind implementat – cu ajutorul circuitelor electronice – în procesor.

Aşadar, o sarcină se poate rezolva cu ajutorul calculatorului numai dacă modul în care se rezolvă poate fi descompus în paşi, pentru a putea fi descris cu ajutorul unui algoritm, deoarece calculatorul este o **maşină algoritmică**.

Dezvoltarea prelucrării automate a informațiilor cu ajutorul calculatorului s-a făcut în două direcții:

- ✓ **dezvoltarea echipamentelor**, astfel încât acestea să fie capabile să stocheze cât mai multă informație, pe care să o prelucreze cu viteză cât mai mare, folosind algoritmi cât mai complecşi;
- ✓ **găsirea de noi algoritmi**, cât mai performanți, pentru rezolvarea problemelor complexe şi **îmbunătățirea tehnicilor de reprezentare şi comunicare** a lor.

1.2. Informatica

Folosirea calculatorului a dus la apariția unei noi ştiințe şi a unui nou domeniu de activitate: **informatica**.

Informatica reprezintă un complex de discipline prin care se asigură prelucrarea rațională a informațiilor prin intermediul maşinilor automate.

Primul calculator electronic a apărut în anul 1946, ca urmare a unei cereri precise din partea armatei americane, care a fost capabilă să finanţeze un proiect atât de costisitor. Apoi, administraţia americană a cumpărat primul calculator non-militar în 1951, pentru recensământul populaţiei. Doi ani mai târziu a fost construit primul calculator destinat unei firme particulare, când General Electric a cumpărat un calculator pentru uzina sa din Louisville. Începând din 1953, firma IBM a început să pătrundă şi ea pe piaţa de calculatoare, prezentându-şi în mediile ştiinţifice propriile calculatoare. Astfel, calculatoarele au început să pătrundă şi în mediile universitare. Dezvoltarea continuă a echipamentelor electronice de calcul a făcut ca, **din 1965, informatica să nu mai fie doar o activitate anexă, ci să se transforme ea însăşi într-o industrie.** În contextul unei creşteri puternice a pieţei, calculatorul a devenit o unealtă folosită în toate domeniile de activitate.

La primele calculatoare electronice, programele erau scrise în cod maşină (binar) sau erau cablate sub formă de circuite electronice. Modificarea unui program şi introducerea unuia nou erau foarte complicate, deoarece însemna introducerea programului bit cu bit. Din necesitatea rezolvării acestei probleme, au apărut primele sisteme de operare şi primele limbaje de programare numite limbaje de nivel înalt: în 1956, limbajul **Fortran** orientat pe calcule tehnico-ştiinţifice, şi în 1960 limbajul **Cobol** orientat pe aplicaţii economice care folosesc puţine operaţii de calcul, dar care manipulează un volum mare de date. Limbajele de programare s-au dezvoltat continuu, pentru a se adapta la noile echipamente hardware, la noile sisteme de operare şi la noile cerinţe ale utilizatorilor – care însemnau de fapt noi sarcini pe care trebuia să le rezolve calculatorul, adică noi algoritmi, orientaţi pe rezolvarea anumitor probleme. În 1971 a fost creat în universităţile elveţiene limbajul **Pascal**, primul **limbaj structurat** (fiecare prelucrare elementară este considerată ca un bloc, iar blocurile pot fi închise – încapsulate – unele în altele). O dată cu apariţia microcalculatoarelor, acest limbaj s-a răspândit foarte mult. Limbajul **Basic** a fost creat în Statele Unite, în 1975, ca un **limbaj interactiv**, şi nu putea fi folosit decât pe microcalculatoare. El permitea abordarea programării şi de către persoane care nu erau specialiste în informatică. În 1971 a fost creat de firma Bell-Telephone limbajul **C**, pentru a permite realizarea sistemului de operare Unix. Este un limbaj foarte performant, care posedă atât conceptele limbajelor structurate de nivel înalt, cât şi conceptele limbajelor de nivel scăzut, care îi permit accesul la hardware. Programele scrise în limbajele apărute recent au crescut productivitatea programatorilor. Limbajele de nivel înalt au pus bazele **ingineriei programării**.

La începutul anilor '60, în mediile universitare au început să se formeze departamente pentru cercetarea şi studierea calculatoarelor. Cu timpul, a apărut o bogată literatură de specialitate, iar cursurile din domeniul informaticii au început să fie orientate pe subdomenii şi să fie gradate pe niveluri de dificultate. Astăzi, informatica este divizată în următoarele nouă **subdomenii**.

1. **Algoritmi şi structuri de date.** Studiază metodele prin care se pot obţine aplicaţii care să prelucreze diferite clase de informaţii, modul în care vor fi reprezentate informaţiile care vor fi prelucrate şi metodele de optimizare a paşilor necesari pentru realizarea aplicaţiilor. Scopul acestui subdomeniu este de a identifica problemele care pot fi descrise cu ajutorul algoritmilor, de a găsi modul în care trebuie procedat pentru a descoperi algoritmul şi metodele de analiză şi comparare a caracteristicilor algoritmilor pentru a obţine algoritmi cât mai eficienţi.

2. **Limbaje de programare.** Studiază notaţiile (limbajele) prin care vor fi reprezentaţi algoritmii şi structurile de date, astfel încât aplicaţia să poată fi prelucrată. Aceste limbaje sunt apropiate de limbajul natural şi pot fi uşor traduse în secvenţe de comenzi pe care să le înţeleagă calculatorul. Scopul acestui subdomeniu este de a găsi noi tehnici de reprezentare şi comunicare a algoritmilor.
3. **Arhitectura calculatoarelor.** Studiază modul în care sunt organizate diferite componente hardware ale calculatorului şi modul în care sunt conectate, pentru a putea obţine un sistem eficient, sigur şi util. Scopul acestui subdomeniu este de a realiza maşini algoritmice cât mai bune folosind cunoştinţele despre algoritmi deja dobândite şi tehnologia existentă.
4. **Sisteme de operare.** Studiază felul în care trebuie să fie organizate programele care controlează şi coordonează toate operaţiile din sistemul de calcul. Scopul acestui subdomeniu este de a face un calculator să rezolve în acelaşi timp mai multe sarcini, fără ca paşii algoritmilor care descriu rezolvarea acestor sarcini să interfereze unii cu alţii, iar atunci când este cazul să se poată realiza comunicarea între diverşi algoritmi.
5. **Ingineria programării.** Studiază metodele prin care poate fi automatizată activitatea de proiectare a aplicaţiilor şi de prelucrare a informaţiilor, astfel încât să se obţină programe corecte, eficiente, fără erori şi uşor de exploatat.
6. **Calcule numerice şi simbolice.** Studiază descrierea fenomenelor din lumea reală prin intermediul formulelor matematice, care pot fi manipulate algebric astfel încât să se obţină modele matematice uşor de descris prin algoritmi. Scopul acestui subdomeniu este de a găsi modele matematice care să permită descrierea şi reprezentarea în calculator a fenomenelor complexe, cum sunt: zborul avioanelor, curenţii marini, traiectoria sateliţilor şi a planetelor, mişcarea particulelor etc.
7. **Sisteme de gestiune a bazelor de date.** Studiază modul în care pot fi organizate cantităţi mari de date ce nu necesită, în prelucrare, calcule matematice complexe. Este cazul informaţiilor prelucrate în procesele economico-sociale, în întreprinderi şi în administraţie. Prelucrarea acestor date trebuie să se facă eficient, fără erori, cu asigurarea securităţii lor.
8. **Inteligenţa artificială.** Studiază modul în care percepe şi raţionează mintea umană, cu scopul de a putea fi automatizate aplicaţii pe care omul le realizează prin metode „inteligente", care sunt dificil de descris cu ajutorul algoritmilor, ca de exemplu înţelegerea unui limbaj, crearea de noi teorii matematice, compunerea muzicii, crearea operelor de artă, luarea deciziilor în urma evaluării unor situaţii complexe (stabilirea unui diagnostic în medicină, mutarea pieselor la jocul de şah etc.).
9. **Animaţie şi robotică.** Studiază metodele prin care pot fi generate şi prelucrate imaginile şi modul în care se poate răspunde unei situaţii din exterior prin acţionarea unui robot.

1.3. Etapele rezolvării unei probleme

Orice prelucrare automată a informaţiilor presupune definirea următorului lanţ:



Din această cauză, pentru orice rezolvare a unei probleme cu ajutorul calculatorului, trebuie parcurse următoarelor **etape**:

1. **analiza problemei;**
2. **elaborarea modului de rezolvare a problemei;**
3. **codificarea într-un limbaj de programare a modului de rezolvare a problemei;**
4. **testarea programului şi corectarea erorilor.**

1. Analiza problemei. Această etapă constă în formularea enunţului problemei din care vor rezulta **specificaţiile** complete şi precise ale programului care va rezolva problema. Aceste specificaţii trebuie să ţină cont de condiţiile concrete de realizare a programului. Specificaţiile sunt:

✓ **Funcţia programului**. Prin ea, se determină ceea ce urmează să realizeze programul.

✓ **Identificarea fluxului de informaţii**. Aceasta presupune identificarea **informaţiilor de intrare** şi, respectiv, a **informaţiilor de ieşire** care vor fi descrise cu ajutorul **datelor**: **date de intrare** şi, respectiv, **date de ieşire**.

Fiecărui tip de informaţie îi corespunde un anumit mod de stocare în mediul de memorare, adică un anumit tip de dată. Între datele prelucrate de un program există diferite relaţii. Modul în care vor fi aranjate aceste date în mediul de memorare depinde de legătura dintre ele.

2. Elaborarea modului de rezolvare a problemei. Această etapă constă în găsirea metodei prin care să se poată rezolva problema. Ea presupune identificarea **prelucrărilor** care se fac asupra datelor de intrare pentru a obţine datele de ieşire. Descrierea acestor prelucrări se face cu ajutorul **algoritmului** de rezolvare a problemei. Această fază este cea mai importantă şi cea mai grea, deoarece presupune definirea logică a unei secvenţe de operaţii pe care să le poată executa calculatorul, astfel încât să se obţină rezultatele dorite.

3. Codificarea într-un limbaj de programare a modului de rezolvare a problemei. Algoritmul de rezolvare a problemei este transpus într-un limbaj de programare ales în conformitate cu specificul problemei care trebuie să fie rezolvată, pentru a fi comunicat calculatorului.

4. Testarea programului şi corectarea erorilor. Pentru testarea programului se va folosi o mulţime de seturi de date de intrare, care trebuie să prevadă toate situaţiile care pot să apară în exploatarea curentă a programului. Testarea constă în executarea repetată a programului, pentru fiecare set de date de intrare. Dacă această mulţime de seturi de date nu este aleasă corect, programul nu va fi testat pe toate traseele algoritmului şi în etapa de exploatare pot apărea erori. În această etapă se pun în evidenţă erorile de sintaxă, erorile de logică şi dacă reprezentarea externă a rezultatelor are aspectul grafic dorit. Erorile de sintaxă apar din scrierea incorectă a instrucţiunilor şi ele vor fi corectate în program. Erorile de logică apar din cauza metodei de rezolvare alese şi ele vor trebui identificate în cadrul algoritmului şi corectate în program.

Aşadar, pentru ca un calculator **să poată produce informaţii**, trebuie ca, la rândul său, **să primească două categorii de informaţii**:

✓ Descrierea modului în care realizează sarcina, adică **algoritmul**, care i se comunică sub forma unui **program**.

✓ Informaţiile de care are nevoie algoritmul ca să realizeze acea sarcină care i se comunică sub formă de **date de intrare.**

Studiu de caz

Scop: exemplificarea etapelor de rezolvare a unei probleme.

Enunţul problemei: *Fiind date două numere reale a şi b, să se rezolve ecuaţia de gradul întâi cu aceşti coeficienţi:* $ax + b = 0$.

În urma analizei problemei, se obţine **specificaţia programului:**

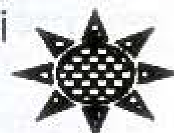
✓ **Funcţia programului.** Dacă pentru ecuaţia de gradul întâi $ax + b = 0$ există o soluţie reală, se calculează; în caz contrar, se afişează un mesaj.

✓ **Informaţiile de intrare** sunt coeficienţii ecuaţiei, iar suportul extern prin care se vor introduce este tastatura. Reprezentarea internă a informaţiei se va face prin **datele de intrare** *a* şi *b*.

✓ **Informaţia de ieşire** va fi soluţia ecuaţiei, dacă există, iar dacă nu există, un mesaj. Suportul extern pe care va fi reprezentată informaţia de ieşire este ecranul monitorului. Reprezentarea internă a soluţiei ecuaţiei se va face prin **data de ieşire** *x*.

Metoda folosită pentru rezolvarea problemei va fi algoritmul matematic de rezolvare a ecuaţiei de gradul întâi.

Pentru testarea programului, se va considera că un set de date de intrare este format de perechea de coeficienţi (a; b), iar o mulţime completă de seturi de date de intrare poate fi {(0; 0), (0; 1.5), (2.5; 1.5)}.



1.4. Algoritmul

Datele de intrare sunt supuse unui proces de prelucrare, pentru a se obţine datele de ieşire. În funcţie de rezultatele care se doresc, prelucrarea datelor este realizată după un anumit algoritm.

> **Algoritmul** reprezintă o **mulţime ordonată şi finită de paşi executabili** prin care se defineşte [redacted] modul în care se poate realiza o anumită [redacted].

Între datele de intrare şi datele de ieşire ale algoritmului există o relaţie bine determinată de însăşi construcţia algoritmului.

În activităţile zilnice întâlnim la tot pasul algoritmi: algoritmul de utilizare a maşinii de spălat rufe sau vase (exprimat prin setul de instrucţiuni din cartea tehnică a maşinii sau de pe capacul maşinii de spălat), algoritmul de înregistrare pe o casetă video (exprimat prin setul de instrucţiuni din cartea tehnică a videorecorderului), algoritmul de interpretare a muzicii (exprimat prin partitură), algoritmul de construire a unui model de avion sau de navă (exprimat prin setul de instrucţiuni care însoţesc piesele care compun modelul), algoritmul de rezolvare a unei probleme matematice (exprimat printr-un set unic de operaţii prin care se descrie modul de rezolvare a oricărei probleme dintr-o categorie de probleme). De fapt, aproape toa-

te acţiunile noastre se desfăşoară după un algoritm bine definit. Un exemplu de algoritm al activităţilor zilnice este o convorbire telefonică:

Pasul 1. *Început.*
Pasul 2. *Mergi la telefon.*
Pasul 3. *Ridică microreceptorul telefonului.*
Pasul 4. *Dacă* ***are ton****, formează numărul de telefon; altfel, pleacă la vecin şi mergi la* **Pasul 10.**
Pasul 5. *Dacă* ***telefonul este ocupat****, închide telefonul şi mergi la* **Pasul 11***; altfel, aşteaptă să răspundă.*
Pasul 6. *Dacă* ***nu răspunde****, pune microreceptorul în furcă şi mergi la* **Pasul 12***; altfel, începi discuţia cu persoana care a răspuns.*
Pasul 7. *Dacă* ***a răspuns persoana căutată****, mergi la* **Pasul 9***; altfel, cere să vină la telefon persoana căutată.*
Pasul 8. *Dacă* ***persoana căutată nu poate să vină la telefon****, mergi la* **Pasul 13***; altfel, aşteaptă să vină la telefon.*
Pasul 9. *Discută la telefon cu persoana căutată şi mergi la* **Pasul 13.**
Pasul 10. *Anunţă la serviciul „Deranjamente telefoane" că ai telefonul defect şi mergi la* **Pasul 14.**
Pasul 11. *Aşteaptă 15 minute şi mergi la* **Pasul 2.**
Pasul 12. *Aşteaptă 1 oră şi mergi la* **Pasul 2.**
Pasul 13. *Închide telefonul.*
Pasul 14. *Terminat.*

Un exemplu de algoritm matematic este rezolvarea ecuaţiei de gradul întâi:

$$a \times z + b = 0$$

unde **a** şi **b** sunt coeficienţii ecuaţiei şi pot lua orice valori din domeniul numerelor reale, iar **z** reprezintă un număr care se calculează şi care poate lua şi el orice valoare reală, astfel încât să fie îndeplinită relaţia definită prin ecuaţie. Algoritmul de rezolvare a ecuaţiei va prezenta un set unic de operaţii, prin care se calculează valoarea lui **z**, oricare ar fi valorile pentru **a** şi **b**:

Pasul 1. *Început.*
Pasul 2. ***Comunică*** *valorile pentru* ***a*** *şi* ***b****.*
Pasul 3. ***Compară*** *a = 0. Dacă este adevărat, execută* **Pasul 4***; altfel, execută* **Pasul 7.**
Pasul 4. ***Compară*** *b = 0. Dacă este adevărat, execută* **Pasul 5***; altfel, execută* **Pasul 6.**
Pasul 5. ***Comunică*** *mesajul "Ecuaţia are o infinitate de soluţii". Mergi la* **Pasul 9.**
Pasul 6. ***Comunică*** *mesajul "Ecuaţia nu are soluţii". Mergi la* **Pasul 9.**
Pasul 7. ***Calculează z = –b/a.***
Pasul 8. ***Comunică*** *valoarea lui* ***z****.*
Pasul 9. *Terminat.*

Numărul de paşi este finit (9 paşi). Toţi paşii reprezintă acţiuni care se pot executa: compară, calculează, comunică. O dată definit acest algoritm, paşii lui se vor executa pentru orice valori ale lui **a** şi **b**, deci algoritmul descrie rezolvarea unei probleme generale. La fiecare executare a algoritmului care descrie o problemă generală va fi tratat un caz particular, adică se rezolvă ecuaţia de gradul întâi pentru va-

lori precizate ale lui **a** şi **b**, ca de exemplu **2×z – 4 = 0 (a = 2, b = –4)** sau **0×z – 4 = 0 (a = 0, b = –4)** sau **0×z – 0 = 0 (a = 0, b = 0)**.

Aşadar, algoritmii au următoarele **proprietăţi:**

- ✓ **Claritatea**. Orice algoritm trebuie să fie precis definit, să prezinte clar toate etapele care trebuie parcurse până la obţinerea soluţiei, fără să formuleze nimic ambiguu.
- ✓ **Finitatea**. Algoritmul trebuie să fie format dintr-un număr finit de paşi, prin executarea cărora să se ajungă la rezolvarea problemei şi obţinerea rezultatelor.
- ✓ **Succesiunea determinată a paşilor**. Paşii care compun algoritmul trebuie executaţi într-o ordine bine determinată. De obicei, ei se execută în ordine secvenţială (ordinea în care au fost scrişi). În cazul în care apare necesitatea schimbării acestei ordini, trebuie să se precizeze clar pasul care urmează să fie executat.
- ✓ **Universalitatea**. Algoritmul trebuie să permită rezolvarea unei clase de probleme, care sunt de acelaşi tip şi care diferă între ele numai prin datele de intrare. El trebuie să ofere posibilitatea de a rezolva orice problemă din acea clasă de probleme.
- ✓ **Realizabilitatea**. Paşii care compun algoritmul trebuie să reprezinte operaţii care se pot executa cu resursele disponibile.
- ✓ **Eficienţa**. Operaţiile care compun algoritmul trebuie alese astfel încât soluţia problemei să fie obţinută după un număr minim de paşi, cu precizia prestabilită sau cu o precizie satisfăcătoare.

Evaluare

Răspundeţi:

1. Ce este un algoritm? Ce sunt paşii algoritmului?
2. Determinaţi algoritmul pentru prepararea unui ceai. Identificaţi proprietăţile unui algoritm, în cazul acestui algoritm.
3. Citiţi o reţetă din cartea de bucate. Determinaţi algoritmul pentru prepararea respectivului produs culinar.
4. Daţi patru exemple de probleme a căror rezolvare nu poate fi descrisă cu ajutorul algoritmului şi patru exemple de probleme a căror rezolvare poate fi descrisă cu ajutorul algoritmului.

2. Datele

2.1. Definiția datelor

Datele sunt obiectele prelucrate de algoritm.

> **Data** este un model de reprezentare a informației, accesibil calculatorului, cu care se poate opera pentru a obține noi informații.

Din punct de vedere logic, data este definită printr-o tripletă:

data elementară = (identificator, valoare, atribute)

Identificatorul datei

Identificatorul datei este un nume format din unul sau mai multe caractere şi este atribuit unei date de către cel care defineşte data, pentru a o putea distinge de alte date şi pentru a putea face referiri la ea în procesul de prelucrare a datelor. De exemplu, *alfa*, *a11* sau *b1_1*.

Fiecare limbaj de programare are implementat diferit conceptul de identificator al datei, practicând constrângeri pentru:

- ✓ **numărul maxim de caractere din nume** (de exemplu, 10 caractere în cazul limbajului de programare Visual Basic şi 64 de caractere în cazul limbajului de programare Pascal);
- ✓ **caracterele acceptate în nume** (de exemplu, majoritatea limbajelor de programare nu acceptă în nume decât cifre, litere şi caracterul linie de subliniere);
- ✓ **caracterul care poate fi folosit la începutul numelui** (de exemplu, marea majoritate a limbajelor de programare nu acceptă ca numele unei date să înceapă cu o cifră).

De aceea, atunci când învățați să definiți şi să manipulați date folosind un anumit limbaj de programare, trebuie să aflați ce constrângeri există pentru identificatorul datei.

Valoarea datei

Valoarea datei reprezintă conținutul zonei de memorie în care este stocată data. Se defineşte ca **domeniu de definiție al datei** mulțimea valorilor pe care le poate lua data în procesul de prelucrare.

Atributele datei

Atributele sunt proprietăți ale datelor care determină modul în care sistemul va trata datele. Cel mai important atribut este **tipul datei**.

> **Tipul datei** defineşte apartenența datei la o anumită clasă de date, căreia îi corespunde un anumit model de reprezentare internă.

Indiferent de tipul de date ales, reprezentarea datei în memoria calculatorului se face printr-un şir de biţi. Pentru a realiza această reprezentare, fiecare limbaj de programare are implementaţi **algoritmi de codificare** care asigură corespondenţa dintre tipul de dată şi şirul de biţi, atât la scrierea datelor, cât şi la citirea lor.

Aşadar, **orice sistem care prelucrează informaţia sub formă de date trebuie să aibă definit clar conceptul de dată.** Definirea conceptului de dată implică definirea următoarelor elemente:

✓ cum poate fi identificată data?
✓ cum va fi reprezentată data în memoria calculatorului?
✓ ce proprietăţi are data?
✓ cum pot fi grupate datele în colecţii de date?

2.1.1. Clasificarea datelor

Clasificarea datelor se poate face folosind mai multe **criterii:**

1. În funcţie de momentul în care se produc în **fluxul de informaţie:**
   ✓ date de intrare
   ✓ date intermediare
   ✓ date de ieşire
2. În funcţie de **valoare:**
   ✓ date variabile
   ✓ date constante
3. În funcţie de **modul de compunere:**
   ✓ date elementare
   ✓ structuri de date
4. În funcţie de **tip:**
   ✓ date numerice
   ✓ date logice
   ✓ date şiruri de caractere

Clasificarea în funcţie de momentul în care se produc

Datele se clasifică în:

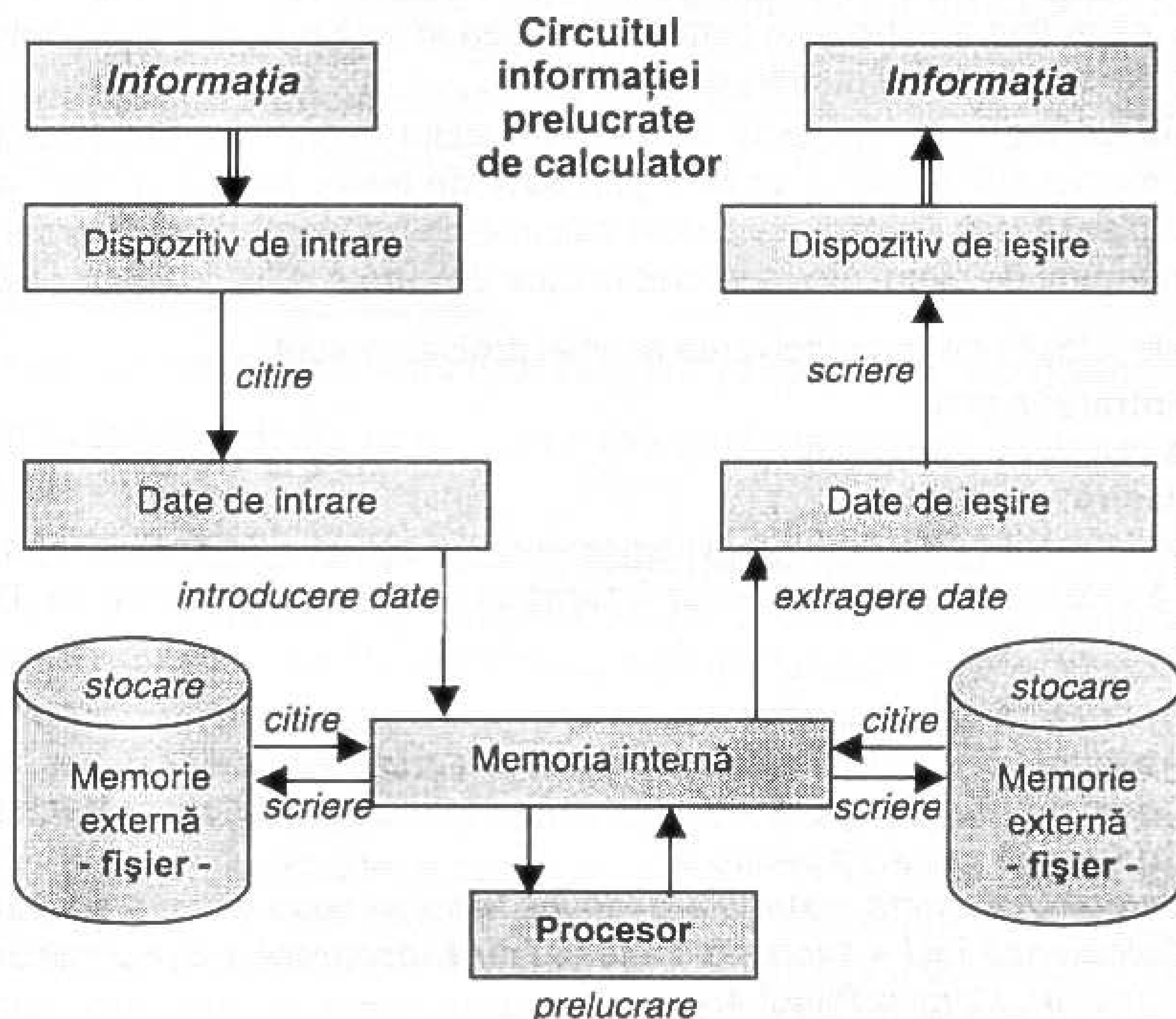
✓ **Date de intrare.** Ele reprezintă datele care urmează să fie prelucrate în cadrul algoritmului. Sunt folosite pentru a descrie diverse evenimente şi sunt informaţii produse în urma realizării unui eveniment, adică evaluări neprelucrate ale acelui eveniment. De exemplu, pot fi date de intrare notele unui elev. Pentru a putea fi prelucrate de procesor, datele sunt introduse în memoria internă a calculatorului. Introducerea datelor se face prin intermediul unor echipamente specializate în **citirea** informaţiei, numite **dispozitive de intrare** (tastatură, scanner, creion optic etc.). **Dispozitivul standard de intrare este *tastatura*.**

✓ **Date de ieşire.** Ele sunt folosite pentru a descrie rezultatele obţinute în urma prelucrărilor din cadrul algoritmului şi furnizează informaţiile pentru care a fost realizat algoritmul, ca de exemplu mediile semestriale şi anuale ale elevului. Datele de ieşire sunt produse de procesor în urma operaţiei de prelucrare şi sunt depuse în memoria internă. Pentru a fi vizualizate de om, ele sunt extrase din memoria internă prin intermediul unor echipamente specializate în **scrierea** infor-

mației, numite **dispozitive de ieşire** (ecran, imprimantă etc.). **Dispozitivul standard de ieşire este *ecranul*.**

✓ **Date intermediare** sau **de manevră**. Ele sunt folosite, în cadrul algoritmului, pentru realizarea unor prelucrări.

În vederea prelucrării, datele pot fi păstrate, temporar, în memoria internă sau în memoria externă (discul flexibil, hard-discul, discul compact etc.). Operația se numeşte **stocarea** datelor.



În memoria externă datele sunt păstrate în fişiere. **Un fişier este o colecție de date organizate ca o singură unitate**. Dacă datele sunt păstrate în **fişiere**, ele vor putea fi folosite ulterior ca date de intrare într-un alt algoritm.

Aşadar, orice rezolvare de problemă începe prin definirea datelor, continuă cu prelucrarea lor în conformitate cu algoritmul folosit şi se termină fie cu afişarea valorii lor, fie cu stocarea lor pe un mediu de memorare, în vederea prelucrării lor, ulterior.

Studiu de caz

Scop: exemplificarea tipurilor de date care pot să apară într-un algoritm.

Enunțul problemei: *Să se calculeze media aritmetică a* **n** *numere întregi introduse de la tastatură.*

În urma analizei problemei se obține **specificația programului:**

✓ **Funcția programului**. Se calculează suma numerelor introduse de la tastatură şi se împarte la numărul de elemente, *n*.

- ✓ **Informațiile de intrare** sunt: numărul de elemente ale mulțimii de numere şi numerele care se citesc. Reprezentarea internă a informației se va face prin **datele de intrare:** ***n*** pentru numărul de valori citite şi ***a*** pentru numărul citit curent.
- ✓ Pentru operațiile executate în cadrul algoritmului, se vor folosi **datele intermediare *suma***, în care se calculează suma numerelor introduse de la tastatură, prin intermediul datei de intrare *a*, şi ***i*** prin care se numără câte valori s-au introdus de la tastatură la un moment dat. Data intermediară *i* este necesară pentru a afla când se termină procesul de introducere a celor *n* numere de la tastatură, ea având funcția unui contor. Valoarea inițială a sumei şi a contorului (înainte de a se citi primul număr) este *0*.
- ✓ **Informația de ieşire** va fi media aritmetică a celor *n* numere. Reprezentarea internă a mediei aritmetice se va face prin **data de ieşire *media***, a cărei valoare se calculează prin împărțirea sumei calculate cu ajutorul datei intermediare *suma* la numărul de elemente memorat în data de intrare *n*.

Aşadar, datele folosite pentru rezolvarea acestei probleme sunt:

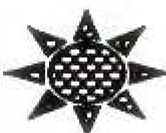
- ✓ **Date de intrare:** *n* şi *a*.
- ✓ **Date intermediare:** *suma* şi *i*.
- ✓ **Date de ieşire:** *media*.

iar algoritmul de rezolvare a problemei va prezenta un set unic de operații prin care se calculează valoarea mediei, oricare ar fi numărul de numere şi valorile lor. Deci:

Pasul 1. *Început.*
Pasul 2. ***Comunică** valoarea pentru **n**.*
Pasul 3. ***Atribuie** valorile inițiale datelor **suma** şi **i**: **suma=0** şi **i=0**.*
Pasul 4. ***Compară** $i \le n$. Dacă este adevărat execută* **Pasul 5**; *altfel, execută* **Pasul 8**.
Pasul 5. ***Comunică** valoarea pentru **a**.*
Pasul 6. ***Calculează suma = suma + a** (adună la sumă noua valoare a lui **a**).*
Pasul 7. ***Calculează i = i + 1** (creşte contorul **i** cu 1, deoarece s-a mai citit un număr **a**). Mergi la* **Pasul 4**.
Pasul 8. ***Calculează media = suma/n.***
Pasul 9. ***Comunică** valoarea datei **media**.*
Pasul 10. *Terminat.*

Observații:

1. Paşii care conțin acțiuni de comunicare folosesc numai date de intrare şi de ieşire, nu şi date intermediare. Datele intermediare apar numai în paşii care conțin acțiuni de calcul, de atribuire sau de comparare.
2. Valoarea datelor de ieşire se calculează în cadrul algoritmului şi se comunică printr-o operație de scriere. Pentru calcularea datelor de ieşire se folosesc date de intrare şi/sau date intermediare. Aceste date trebuie să aibă o valoare, înainte de a fi folosite în paşii care conțin acțiuni de calcul. Valoarea datelor de intrare este comunicată prin operația de citire, în acest caz, de la tastatură. Datelor intermediare li se atribuie o valoare inițială în cadrul algoritmului.
3. **Orice nouă operație de citire executată cu o dată de intrare distruge vechea valoare memorată în dată.**



Clasificarea în funcție de valoare

Datele se clasifică în:

✓ **Date variabile** sau **variabile de memorie**. Pe parcursul procesului de prelucrare, valoarea acestor date se poate modifica, în limitele domeniului de definiție. În timpul execuției programului, ele pot avea o valore inițială, mai multe valori intermediare şi o valoare finală. În exemplul precedent, pentru calcularea mediei a *n* numere introduse de la tastatură, data *suma* are o valoare inițială *0*, mai multe valori intermediare, câte una pentru fiecare operație de citire a unui număr *a*, şi o valoare finală, obținută după ce s-au citit toate cele *n* numere.

✓ **Date constante** sau **constante**. Pe tot parcursul procesului de prelucrare, data îşi va păstra aceeaşi valoare din domeniul de definiție al datei.

Studiu de caz

Scop: exemplificarea tipurilor de date care pot să apară într-un algoritm.

Enunțul problemei 1: *Să se calculeze aria pentru* **n** *cercuri, fiecare cerc având o rază precizată*, **r**.

În urma analizei problemei se obține **specificația programului**:

✓ **Funcția programului**. Se calculează aria pentru *n* cercuri, folosind formula matematică $aria = \pi \times r^2$, unde *r* este raza unuia dintre cele *n* cercuri.

✓ **Informațiile de intrare** sunt: numărul de cercuri, razele cercurilor şi valoarea numărului π. Reprezentarea internă a informației se va face prin **datele de intrare**: ***n*** pentru numărul de cercuri, ***r*** pentru raza unui cerc, şi ***pi*** pentru numărul π.

✓ Pentru operațiile executate în cadrul algoritmului, se va folosi **data intermediară *i***, care reprezintă numărul cercului pentru care se calculează aria. Această dată intermediară este necesară pentru a afla când se termină procesul de calculare a ariei pentru cele *n* cercuri şi are valoarea inițială *0* (nu s-a calculat aria nici unui cerc).

✓ **Informațiile de ieşire** vor fi ariile celor *n* cercuri. Reprezentarea internă a ariei unui cerc se va face prin **data de ieşire *aria***, a cărei valoare se calculează prin formula matematică. $aria = \pi \times r^2$.

Algoritmul de rezolvare a problemei va prezenta un set unic de operații prin care se calculează valoarea ariei, oricare ar fi numărul de cercuri şi valoarea razelor lor:

Pasul 1. *Început.*
Pasul 2. ***Comunică*** *valoarea pentru* ***n***.
Pasul 3. ***Atribuie*** *valoarea inițială contorului* ***i***: ***i = 0***.
Pasul 4. ***Compară*** ***i*** $\le$ ***n***. *Dacă este adevărat, execută* **Pasul 5**; *altfel, execută* **Pasul 9**.
Pasul 5. ***Comunică*** *valoarea pentru* ***r***.
Pasul 6. ***Calculează aria = pi*** $\times$ ***r***^{2}.
Pasul 7. ***Comunică*** *valoarea* ***ariei*** *pentru cercul* ***i***.
Pasul 8. ***Calculează i = i + 1*** *(creşte contorul* ***i*** *cu 1, deoarece s-a citit raza unui cerc r). Mergi la* **Pasul 4**.
Pasul 9. *Terminat.*

Datele *n*, *i*, *aria* şi *r* sunt date variabile. Valoarea datei *n* se modifică pentru fiecare execuție a algoritmului (corespunde valorii introduse de la tastatură). Valoarea da-

telor *i*, *r* şi *aria* se modifică în funcţie de numărul cercului pentru care se citeşte raza şi pentru care se calculează aria în timpul execuţiei algoritmului. Data *pi* este o dată constantă. Indiferent de datele de intrare cu care se execută algoritmul, valoarea ei este aceeaşi şi corespunde valorii numărului π.

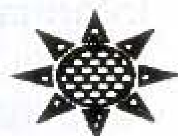
Enunţul problemei 2: *Să se afişeze numerele pare care au două cifre.*

În urma analizei problemei se obţine **specificaţia programului:**

- ✓ **Funcţia programului**. Se generează numerele pare dintr-un interval [*a*,*b*].
- ✓ **Informaţiile de intrare** sunt limitele intervalului. Deoarece din enunţul problemei rezultă că ***a*** *= 10* (primul număr par cu două cifre) şi ***b*** *= 98* (ultimul număr par cu două cifre), reprezentarea internă a informaţiei se va face prin **datele de intrare constante**: *10* şi *98*. Deoarece aceste date au valori constante, ele nu trebuie citite de la tastatură.
- ✓ **Informaţiile de ieşire** vor fi numerele pare din intervalul precizat. Reprezentarea internă a unui număr par se va face prin **data de ieşire** *n* a cărei valoare se calculează prin incrementarea cu *2* a valorii anterioare: *n = n + 2*.

Algoritmul de rezolvare a problemei va fi:

Pasul 1. *Început.*
Pasul 2. ***Atribuie** valoarea iniţială numărului **n**: **n = 10**.*
Pasul 3. ***Compară n ≤ 98**. Dacă este adevărat, execută* **Pasul 4**; *altfel, execută* **Pasul 6.**
Pasul 4. ***Calculează n=n+2.***
Pasul 5. ***Comunică** valoarea lui **n**. Mergi la* **Pasul 3.**
Pasul 6. *Terminat.*



Clasificarea în funcţie de modul de compunere

Datele se clasifică în:

- ✓ **Date simple** sau **date elementare**. Sunt date independente unele de altele din punct de vedere al reprezentării lor în memorie. Chiar dacă ele pot depinde din punct de vedere logic (valoarea unei date este dependentă de valoarea altei date), ele nu depind din punct de vedere fizic (localizarea unei date pe suportul de memorare nu se face în funcţie de locaţia pe suport a unei alte date). În calcularea ariei unui cerc, datele *n*, *i*, *r* şi *aria* sunt date elementare.
- ✓ **Date compuse** sau **structuri de date**. Sunt colecţii de date între care există anumite relaţii. Fiecare componentă a structurii are o anumită poziţie în cadrul structurii, iar toate componentele formează un întreg, astfel încât prelucrarea se poate face atât la nivelul structurii de date (care poate fi considerată o entitate de sine stătătoare), cât şi la nivelul fiecărei componente. Pentru fiecare tip de structură de date, în limbajul de programare trebuie să fie definiţi algoritmi de localizare a componentelor în cadrul structurii de date. Între componentele structurii există şi legături de conţinut, adică întregul ansamblu de date din colecţie poate caracteriza un obiect, o persoană, un fenomen, un proces etc. De exemplu, o colecţie cu 12 elemente, în care se memorează valorile lunare ale unui contor electric. Structura de date caracterizează în acest caz un proces: consumul lunar de energie electrică. Aşadar orice obiect, proces sau fenomen din lumea reală poate

fi caracterizat printr-o listă de **proprietăţi**. Valorile proprietăţilor din listă pot fi reprezentate în calculator (lumea virtuală) sub forma unei colecţii de date.

Să ne închipuim că într-un algoritm trebuie reprezentată o clasă: profesorul şi cei *n* elevi. Ei reprezintă datele pe care le prelucrează algoritmul. Profesorului îi va corespunde o dată elementară, iar grupului de elevi o structură de date. Locul ocupat în clasă de fiecare dintre ei reprezintă zona de memorie alocată datei: catedra este zona de memorie alocată profesorului, iar grupul de bănci, zona de memorie alocată elevilor. Cele două zone sunt independente. În schimb, în cadrul zonei de bănci (zona de memorie a structurii de date), fiecărui element de structură (elevul) i se alocă un loc într-o bancă, poziţia sa putând fi identificată după numărul băncii. Dacă pentru grupul de elevi nu s-ar folosi o structură de date, ci date elementare, fiecărei date elementare va trebui să îi dăm un nume, iar algoritmul ar fi foarte greu de scris. În primul rând, nu ştim câte date elementare să folosim pentru elevi. Algoritmul trebuie să fie general, deci să funcţioneze şi pentru o clasă cu 25 de elevi, dar şi pentru o clasă cu 30 de elevi. În timpul anului şcolar, poate să vină în clasă un elev nou, sau poate să plece din clasă un elev. Ce se întâmplă în acest caz cu datele elementare, deoarece la o execuţie a algoritmului, atunci când vine un elev nou în clasă ar trebui să apară o nouă dată elementară, iar la o altă execuţie a algoritmului, atunci când pleacă un elev din clasă, trebuie să dispară o dată elementară? Soluţia este de a folosi o **colecţie de date**. Colecţia va avea atâtea elemente câte bănci sunt în clasă. Se va folosi un singur nume de dată, care se va atribui colecţiei, fiecare element identificându-se apoi după numărul băncii.

Studiu de caz

Scop: exemplificarea modului de compunere a datelor.

Enunţul problemei 1: *Să se calculeze media aritmetică a* **n** *numere introduse de la tastatură.*

Enunţul problemei 2: *Se introduc* **n** *numere de la tastatură. Să se afişeze aceste numere, ordonate crescător.*

Pentru problema 1, se pot folosi numai date elementare, deoarece după citirea unui număr prin intermediul datei *a*, el se prelucrează imediat (se adună la sumă) şi variabila de memorie va putea fi refolosită apoi pentru citirea unui alt număr.

Pentru problema 2, nu se poate folosi decât o colecţie de date, deoarece, pentru aranjarea într-o anumită ordine a celor *n* numere citite de la tastatură trebuie să se păstreze în memorie toate aceste date, pentru a se putea compara între ele în vederea ordonării.



2.1.2. Tipul datei

Tipul datei determină:

- ✓ **dimensiunea zonei de memorie** alocată datei (se măsoară în octeţi);
- ✓ **operatorii** care pot fi aplicaţi pe acea dată;
- ✓ modul în care data este **reprezentată în memoria internă** (metoda de codificare în binar a valorii datei).

Tipul datei este definit prin dubletul (**V**, O), unde:

V = domeniul de definiţie intern al datei;

O= mulţimea operatorilor care se pot aplica pe mulţimea de valori ale datei.

Limbajele de programare acceptă următoarele **tipuri de date**:

- **tipul numeric**
- **tipul logic**
- **tipul şir de caractere**

Tipul numeric

Tipul numeric a fost implementat pentru reprezentarea numerelor întregi sau cu zecimale, pozitive sau negative, şi pentru a realiza majoritatea operaţiilor matematice întâlnite în practică. Pentru tipul numeric există subtipurile **real** şi **întreg**.

Deci: **V** = **R** (mulţimea numerelor reale) sau **I** (mulţimea numerelor întregi)

$O = \mathcal{M}^1 \cup \mathcal{R}^2$

Constantele de tip numeric se reprezintă prin numere cu semn sau fără semn, folosindu-se punctul pentru separarea părţii întregi de partea zecimală: 2; –0.15; 3.175; 20.0.

Tipul logic

Tipul logic, sau boolean, a fost implementat pentru reprezentarea datelor care nu pot lua decât două valori: adevărat (*true*), pe care o notăm cu *T*, sau fals (*false*), pe care o notăm cu *F*.

Deci: **V** = **L** (mulţimea valorilor logice) = {*T*, *F*};

$O = \mathcal{L}^3$

Tipul şir de caractere

Tipul şir de caractere a fost implementat pentru reprezentarea unei mulţimi ordonate de caractere care este tratată ca un tot unitar.

Deci: **V** = {$\mathbf{P_C}$} (mulţimea părţilor mulţimii $\mathbf{C}^4$);

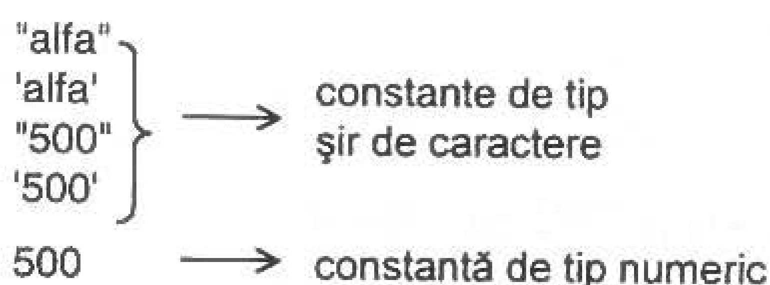
$O = \mathcal{C}^5 \cup \mathcal{R}$

În memoria internă, fiecare caracter din şir se reprezintă prin codul său ASCII. Constantele de tip şir de caractere se specifică prin mulţimea ordonată de caractere care compun şirul, delimitată în funcţie de limbajul de programare, de anumite semne speciale: apostrofurile ('Bună ziua') sau ghilimelele ("Bună ziua").

alfa ⟶ identificator de dată elementară

"alfa'
'alfa" } ⟶ construcţii greşite

[1] Mulţimea operatorilor matematici.
[2] Mulţimea operatorilor relaţionali (de comparare).
[3] Mulţimea operatorilor logici.
[4] Mulţimea caracterelor care este formată din litere, cifre şi semne speciale.
[5] Mulţimea operatorilor de concatenare.



Constanta de tip numeric 500 este diferită de constanta de tip şir de caractere "500", atât din punct de vedere al modului de reprezentare în memoria internă a calculatorului, cât şi din punct de vedere al operatorilor acceptaţi. De exemplu, asupra constantei numerice se pot aplica operatori matematici şi relaţionali, iar asupra constantei de tip şir de caractere, operatori de concatenare şi relaţionali. Constanta de tip numeric este reprezentată în memoria internă prin conversia în binar a numărului, iar constanta de tip şir de caractere este reprezentată prin conversia fiecărui caracter din şir în 8 cifre binare corespunzătoare codului ASCII al caracterului respectiv.

2.2. Operatorii

Operatorii sunt caractere speciale (*, /, >, = etc.) sau cuvinte cheie[6] (**mod, and** etc.) prin intermediul cărora se reprezintă operaţiile care se efectuează în cadrul unui algoritm. Fiecare limbaj de programare are implementat propriul set de operatori. În acest capitol vor fi prezentaţi operatorii care se folosesc în cadrul unui algoritm.

Asupra operanzilor dintr-o expresie puteţi aplica următorii operatori:

- **operatorul de atribuire,**
- **operatorii matematici,**
- **operatorul de concatenare a şirurilor de caractere,**
- **operatorii relaţionali,**
- **operatorii logici.**

Operatorii pot fi aplicaţi numai pe anumite tipuri de operanzi, producând rezultate de un anumit tip. Dacă notăm cu a şi b operanzii asupra cărora se aplică operatorul, cu ☼ operatorul şi cu c rezultatul:

$$a \text{ ☼ } b = c$$

atunci relaţia între a, b, ☼ şi c este dată de următorul tabel:

Tipul operanzilor a şi b	Tipul operatorului ☼	Tipul rezultatului c
numeric	matematic ($\mathcal{M}$)	numeric
numeric	relaţional ($\mathcal{R}$)	logic
şir de caractere	concatenare ($\mathcal{C}$)	şir de caractere
şir de caractere	relaţional ($\mathcal{R}$)	logic
logic	logic ($\mathcal{L}$)	logic

[6] Cuvinte rezervate, care nu mai pot fi folosite ca identificatori pentru date, deoarece deja ele au un înţeles bine stabilit pentru limbajul de programare.

Operatorii matematici

𝓜 = {+, -, *, /, **|^[7], **mod**, **div**}

Se aplică pe date de tip numeric şi furnizează un rezultat de tip numeric.

Operator	Semnificație	Exemplu
+ (adunare)	Adună matematic cei doi operanzi.	5+2=7
- (scădere)	Scade al doilea operand din primul operand.	7-3=4
/ (împărțire reală)	Împarte primul operand la al doilea operand.	7/2=3.5
* (înmulțire)	Înmulţeşte cei doi operanzi.	2*4=8
**|^ (ridicare la putere)	Ridică primul operand la puterea furnizată de cel de al doilea operand.	2**3=8
mod (modulo)	Calculează restul împărțirii primului operand la al doilea operand.	19 mod 4 = 3
div (împărțire întreagă)	Calculează câtul împărțirii primului operand la al doilea operand.	19 div 4 = 4

Operatorii relaționali (de comparație)

𝓡 = {>, >=, <, <=, =, #|<>}

Se aplică pe operanzi de tip numeric sau de tip şir de caractere şi furnizează un rezultat de tip logic.

Operator	Semnificație	Exemplu
= (egalitate)	Rezultatul este *T* dacă cei doi operanzi sunt egali.	(5=5)=*T* (5=7)=*F*
<>|# (diferit)	Rezultatul este *T* dacă cei doi operanzi sunt diferiți.	(5<>5)=*F* (5<>7)=*T*
< (mai mic)	Rezultatul este *T* dacă primul operand este mai mic decât al doilea operand.	(5<7)=*T* (7<5)=*F*
> (mai mare)	Rezultatul este *T* dacă primul operand este mai mare decât al doilea operand.	(7>5)=*T* (5>7)=*F*
<= (mai mic sau egal)	Rezultatul este *T* dacă primul operand este mai mic sau cel mult egal față de al doilea operand.	(5<=5)=*T* (7<=5)=*F*
>= (mai mare sau egal)	Rezultatul este *T* dacă primul operand este mai mare sau cel mult egal față de al doilea operand.	(7>=5)=*T* (5>=7)=*F*

Un operator relațional aplicat pe două şiruri de caractere realizează compararea celor două şiruri de caractere.

Compararea a două caractere este posibilă prin compararea numerică a codurilor ASCII ale celor două caractere. Astfel, codul ASCII al caracterului **d** este 100, iar al caracterului **D** este 68. Deci, caracterul **d** este mai mare decât caracterul **D**.

[7] **Convenție de notare:** Semnul | plasat între două elemente are semnificația conjuncției *sau*, adică pentru operația de ridicare la putere pot fi folosite simbolurile ** *sau* simbolul ^.

Pentru compararea celor două caractere se va scrie "d">"D", iar în urma executării operației de comparare se va produce rezultatul logic *T*, deoarece operația de comparare se execută între cele două valori numerice (100>68).

Compararea a două şiruri de caractere se face prin compararea codului ASCII al caracterelor din aceeaşi poziţie a fiecărui şir. Dacă cele două şiruri nu au aceeaşi lungime, şirul cu lungime mai mică este completat la sfârşit, până la egalarea lungimilor, cu caracterul care are codul ASCII 0. Operaţia de comparare începe cu prima poziţie din şir şi continuă cu următoarele poziţii, numai dacă poziţiile anterioare sunt identice în ambele şiruri. De exemplu, şirul de caractere **Idee** este mai mare decât şirul de caractere **IDEE**, deoarece, în poziţia a doua, caracterele din cele două şiruri nu mai sunt identice, iar codul ASCII al caracterului **d** este mai mare decât codul ASCII al caracterului **D**. Operaţia de comparare se opreşte după cel de al doilea caracter şi nu mai contează codurile caracterelor din poziţiile următoare. Pentru compararea celor două şiruri de caractere se va scrie "Idee">"IDEE", iar în urma executării operaţiei de comparare se va produce rezultatul logic *T*.

Operatorul de concatenare

$\mathcal{C}$ = {+}

Se aplică pe operanzi de tip şir de caractere şi furnizează un rezultat de tip şir de caractere.

Operator	Semnificaţie	Exemplu
+ (concatenare)	Reuneşte două şiruri de caractere într-un singur şir de caractere.	"Buna "+"ziua"= "Buna ziua"

Operatorii logici

$\mathcal{L}$= {and, or, not}

Se aplică numai pe operanzi de tip logic şi furnizează un rezultat de tip logic.

Operator	Semnificaţie	*a*	not *a*
not (negare)	Schimbă valoarea unui operand cu complementul său: *T* cu *F* şi *F* cu *T*.	*T*	*F*
		F	*T*

Operator	Semnificaţie	*a*	*b*	*a* and *b*	*a* or *b*
and („şi" logic)	Rezultatul este *T* dacă ambii operanzi au valoarea *T*, altfel este *F*.	*T*	*T*	*T*	*T*
		T	*F*	*F*	*T*
or („sau" logic)	Rezultatul este *T* dacă cel puţin unul dintre operanzi are valoarea *T*, altfel este *F*.	*F*	*T*	*F*	*T*
		F	*F*	*F*	*F*

Operatorii logici sunt foarte utili atunci când construiţi expresii logice care descriu anumite condiţii ce vor fi testate, urmând ca, în funcţie de rezultat, să se execute anumite operaţii. Dacă expresiile logice sunt prea complexe, le puteţi simplifica folosind următoarele relaţii:

not (a and b) = (not a) or (not b)

not (a or b) = (not a) and (not b)

Operatorul de atribuire

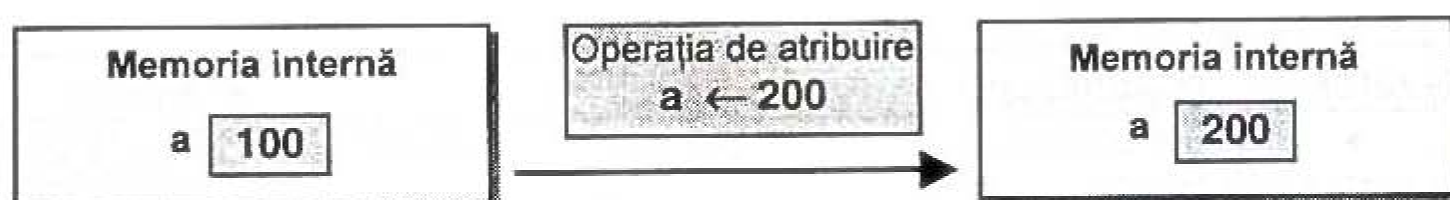
Prin acest operator puteți atribui o anumită valoare unei date:

nume ← expresie

Operația de atribuire se desfășoară în două etape: mai întâi se evaluează expresia, după care valoarea obținută se atribuie datei identificate prin *nume*. Rezultatul evaluării expresiei trebuie să fie de același tip cu data identificată prin *nume*.

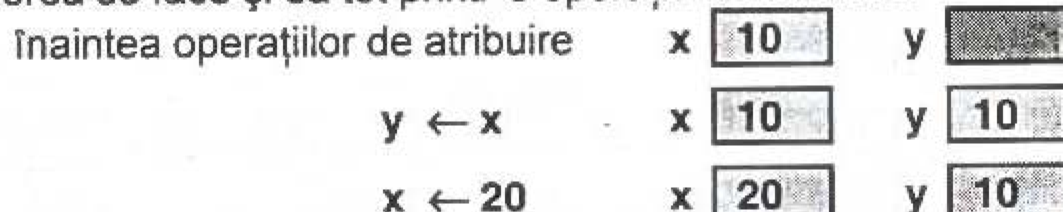
Exemplu	Semnificație
n ←100	Datei de tip numeric *n* i se atribuie valoarea 100.
n ← n+200	Se evaluează expresia adunându-se 200 la valoarea datei *n*. Valoarea obținută în urma evaluării este 300 și se atribuie ca valoare nouă datei *n*.
text ← "șir de caractere"	Datei de tip șir de caractere *text* i se atribuie valoarea "șir de caractere".

Prin operația de atribuire, în zona de memorie alocată datei se scrie noua valoare a datei, vechea valoare pierzându-se.



înainte de operația de atribuire — după operația de atribuire

Dacă vreți să păstrați și vechea valoare, va trebui ca, înainte de operația de atribuire, să **o salvați prin copiere** într-o altă variabilă de memorie (dată variabilă) de același tip. Copierea se face și ea tot printr-o operație de atribuire.



Observație: Operația de atribuire se mai folosește, în cadrul unui algoritm, pentru executarea următoarelor operații:

- ✓ **inițializarea** valorii unor variabile de memorie
- ✓ **calculul iterativ** al valorii unor variabile de memorie.

Calculul iterativ al valorii unei variabile de memorie înseamnă să-i atribuiți acelei variabile de memorie valoarea unei expresii în care unul dintre operanzi este chiar numele acelei variabile de memorie. Tipurile de variabile de memorie cel mai des folosite (din punct de vedere al informației pe care o reprezintă) sunt:

- ✓ **Contorul** sau **numărătorul**. Este o variabilă de memorie care se folosește pentru a număra anumite cazuri care pot să apară. Înainte de a începe numărătoarea, contorul se inițializează cu valoarea 0 (nu a fost evidențiat încă nici un caz). În timpul procesului de prelucrare, atunci când se întâlnește un caz, valoarea

contorului se calculează iterativ prin **incrementare** cu 1 (se creşte valoarea contorului cu 1). De exemplu, dacă folosim variabila de memorie **k** pentru contor, **iniţializarea** sa se va face cu operaţia de atribuire: **k ← 0**, iar **calculul iterativ**, cu operaţia de atribuire: **k ← k+1**.

- ✓ **Suma**. Este o variabilă de memorie care se foloseşte pentru calculul iterativ al unei sume (adunarea, la valoarea anterioară, a unei noi valori). Înainte de a începe calcularea ei, suma se iniţializează cu valoarea 0 (elementul neutru pentru operaţia de adunare, şi înseamnă că nu a fost adunată nici o valoare la sumă). De exemplu, dacă folosim variabila de memorie **s** pentru a calcula suma mai multor valori citite de la tastatură prin intermediul unei variabile de memorie **a**, **iniţializarea** sumei se face cu operaţa de atribuire **s ← 0**, iar **calculul iterativ** cu operaţia de atribuire: **s ← s+a**.
- ✓ **Produsul**. Este o variabilă de memorie care se foloseşte pentru calculul iterativ al unui produs (înmulţirea valorii anterioare cu o nouă valoare). Înainte de a începe calcularea lui, produsul se iniţializează cu valoarea 1 (elementul neutru pentru operaţia de înmulţire, şi înseamnă că nu a fost înmulţită nici o valoare cu produsul). De exemplu, dacă folosim variabila de memorie **p** pentru a calcula produsul mai multor valori citite de la tastatură prin intermediul unei variabile de memorie **a**, **iniţializarea** produsului se face cu operaţia de atribuire **p ← 1**, iar **calculul iterativ** cu operaţia de atribuire: **p ← p*a**.

Studiu de caz

Scop. Exemplificarea modului în care folosiţi operatorul de atribuire pentru iniţializare şi calcul iterativ.

Enunţul problemei: *Se introduc* n *numere întregi de la tastatură. Să se numere câte sunt pare şi să se calculeze suma şi produsul lor.*

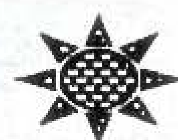
În urma analizei problemei se obţine **specificaţia programului**:

- ✓ **Funcţia programului**. Se numără şi se calculează suma şi produsul numerelor pare dintr-o mulţime de *n* numere întregi. Din enunţul problemei rezultă că în procesul de calcul se folosesc numai numere pare. Pentru a evidenţia acest caz (apariţia unui număr par în mulţimea de numere citite), se foloseşte proprietatea numerelor pare: restul împărţirii unui număr par la 2 este 0, care se exprimă prin expresia, cu rezultat de tip logic, ***(a mod 2)=0***. Dacă această expresie are valoarea *T*, numărul este par.
- ✓ **Informaţiile de intrare** sunt numărul de elemente ale mulţimii de numere întregi şi numerele citite. Reprezentarea internă a informaţiei se va face prin **datele de intrare: *n*** pentru numărul de elemente ale mulţimii şi ***a*** pentru numărul citit curent.
- ✓ Pentru operaţiile executate în cadrul algoritmului, se va folosi **data intermediară *i***, prin care se numără câte numere s-au introdus de la tastatură la un moment dat. Data intermediară *i* este necesară pentru a afla când se termină procesul de citire a celor *n* numere, având funcţia unui contor. Valoarea iniţială a contorului *i* (înainte de a se citi primul număr) este *0*.

✓ **Informațiile de ieșire** vor fi numărul de valori pare, suma și produsul lor. Reprezentarea lor internă se va face prin **datele de ieșire**: ***k*** pentru numărul de valori (contorul numerelor pare), ***s*** pentru suma lor și ***p*** pentru produsul lor. Valoarea inițială (înainte de a se citi primul număr) a contorului *k* și a sumei *s* este *0*, iar a produsului *p* este 1.

Algoritmul de rezolvare a problemei va fi:

Pasul 1. *Început.*
Pasul 2. ***Comunică*** *valoarea pentru* ***n.***
Pasul 3. ***Atribuie*** *valoarea inițială pentru contorul numerelor citite,* ***i: i ← 0;***
Pasul 4. ***Atribuie*** *valoarea inițială pentru contorul numerelor pare,* ***k: k ← 0;***
Pasul 5. ***Atribuie*** *valoarea inițială pentru suma numerelor pare,* ***s: s ← 0;***
Pasul 6. ***Atribuie*** *valoarea inițială pentru produsul numerelor pare,* ***p: p ← 1;***
Pasul 7. ***Compară i<=n.*** *Dacă este adevărat, execută* **Pasul 8**; *altfel, execută* **Pasul 14**.
Pasul 8. ***Comunică*** *valoarea pentru* ***a.***
Pasul 9. ***Compară (a mod 2) = 0.*** *Dacă este adevărat, execută* **Pasul 10**; *altfel, execută* **Pasul 13**.
Pasul 10. ***Actualizează*** *valoarea pentru* ***k: k ← k+1;***
Pasul 11. ***Actualizează*** *valoarea pentru* ***s: s ← s+a;***
Pasul 12. ***Actualizează*** *valoarea pentru* ***p: p ← p*a;***
Pasul 13. ***Actualizează*** *valoarea pentru* ***i: i ← i+1.*** *Mergi la* **Pasul 7.**
Pasul 14. ***Comunică*** *valorile pentru* ***k, s, p.***
Pasul 15. *Terminat.*



2.3. Expresiile

Expresia este o combinație validă de operatori și operanzi.

Operanzii pot fi nume de date, constante de tip numeric sau șir de caractere și funcții care, în urma evaluării, furnizează un singur rezultat. În funcție de tipul operanzilor și al operatorilor, rezultatul expresiei poate fi de tip numeric, șir de caractere sau logic.

Într-o expresie, operatorii se folosesc într-un anumit scop:

✓ **operatorii matematici**, pentru efectuarea calculelor;
✓ **operatorii relaționali** și **logici**, pentru efectuarea de comparații în vederea luării unor decizii în cadrul algoritmului;
✓ **operatorul de atribuire**, pentru manipularea datelor.

Expresia este validă numai dacă operatorii care leagă operanzii corespund tipului operanzilor pe care îi leagă. De exemplu:

✓ Expresia E←22+75 este o expresie validă deoarece operatorul matematic **+** leagă doi operanzi exprimați prin două constante de tip numeric.
✓ Expresia E←"Ana"+"-Maria" este o expresie validă, deoarece operatorul de concatenare **+** leagă doi operanzi exprimați prin două constante de tip șir de caractere.
✓ Expresia E←22+"ani" nu este o expresie validă, deoarece operatorul **+** leagă doi operanzi exprimați prin două constante de tipuri diferite: numeric și, respec-

tiv, şir de caractere. El nu poate fi interpretat nici ca operator matematic, nici ca operator de concatenare.

Dacă, de exemplu, într-o expresie trebuie să se calculeze radicalul dintr-o dată de tip numeric, nu se poate folosi nici un operator matematic. În acest caz, limbajul de programare în care se lucrează pune la dispoziţie o **funcţie**.

Funcţia este o prelucrare predefinită de autorii limbajului de programare, care se poate folosi în cadrul unei expresii la fel ca un operand, deoarece ea este de fapt un program care, în urma execuţiei, furnizează o valoare chiar prin numele ei. Atunci când se evaluează expresia, funcţia este înlocuită cu valoarea returnată în urma executării ei. Funcţia se identifică printr-un nume, iar pentru apelarea ei se scrie numele funcţiei urmat de o listă de parametri precizaţi între paranteze rotunde. Parametrii sunt valorile cu care se execută funcţia la acel apel.

De exemplu, funcţia **sin(*x*)** returnează valoarea funcţiei trigonometrice *sinus* pentru numărul *x*. Numele funcţiei este **sin** iar parametrul *x* este de tip numeric. Dacă se evaluează funcţia *sin(x)*, iar *x* are valoarea 2.5, în urma execuţiei programului asociat, funcţia va furniza valoarea 0.598.

Pentru a evalua o expresie, calculatorul va executa într-o anumită ordine operaţiile pe care le-aţi scris. Această ordine este dată de:

- ✓ **Precedenţa operatorilor**. Este ordinea în care se execută operaţiile definite de operatori. Ea determină nivelul de prioritate al operatorilor.
- ✓ **Asociativitatea operatorilor**. Este ordinea în care se evaluează operatorii cu acelaşi nivel de prioritate.

Fiecare limbaj de programare are implementată **o tabelă de precedenţă** (tabela nivelurilor de prioritate), şi pentru fiecare nivel de prioritate o anumită asociativitate. Într-un algoritm vom folosi următoarea tabelă de precedenţă:

1. Se evaluează funcţiile.
2. Se evaluează operatorii matematici. Operatorii matematici au niveluri de prioritate diferite. Operatorul ** are nivel de prioritate 1, fiind primul care se evaluează, operatorii *, /, **div** şi **mod** au nivelul de prioritate 2, iar operatorii **+** şi **–** au nivelul de prioritate 3.
3. Se evaluează operatorii de concatenare.
4. Se evaluează operatorii relaţionali. Toţi operatorii relaţionali au acelaşi nivel de prioritate.
5. Se evaluează operatorii logici. Ordinea de prioritate a operatorilor logici este **not**, **and** şi **or**, primul fiind cel mai prioritar.

Asociativitatea operatorilor este de la stânga la dreapta, adică toţi operatorii care au acelaşi nivel de prioritate se evaluează în ordine, de la stânga la dreapta.

De exemplu, următoarele expresii vor fi evaluate astfel:

- ✓ ***5*7*4 + 4/2 - 2**3mod2/4*** = 5*7*4 + 4/2 - 8mod2/4 = 35*4 + 4/2 - 8mod2/4 = 140 + 4/2 - 8mod2/4 = 140 + 2 - 8mod2/4 = 140 + 2 - 0/4 = 140 + 2 - 0 = 142 - 0 = ***142***
- ✓ ***3<=5 or 7>8*** = *t* or 7>8 = *t* or *f* = ***t***
- ✓ ***3<=5 and not 7>8*** = *t* and not 7>8 = *t* and not *f* = *t* and *t* = ***t***

Ordinea operaţiilor care se vor executa pentru a evalua expresia *e*:

e ← a*bd>c or a/b=d and a+b<d or not a>c**

este:

e1 ← b**d (operator aritmetic de prioritate 1)
e2 ←a*e1 (operator aritmetic de prioritate 2)
e3 ← a/b (operator aritmetic de prioritate 2)
e4 ← a + b (operator aritmetic de prioritate 3)
e5 ← e2 > c (operator relaţional)
e6 ← e3=d (operator relaţional)
e7 ← e4<d (operator relaţional)
e8 ← a>c (operator relaţional)
e9 ← not e8 (operator logic)
e10 ← e6 and e7 (operator logic)
e11 ← e5 or e10 (operator logic)
e ← e11 or e9 (operator logic)

Ordinea de evaluare a unei expresii poate fi schimbată prin folosirea parantezelor. Se folosesc numai paranteze rotunde, care pot fi şi imbricate.

Atenţie! Numărul de paranteze deschise trebuie să fie egal cu numărul de paranteze închise.

De exemplu, următoarea expresie aritmetică prin care se calculează valoarea lui *E*:

$$E = \frac{a+b}{c} + \frac{b+d}{c+a} + \frac{a+c}{(a+b)^3}$$

va fi scrisă astfel:

E ← (a+b)/c + (b+d)/(c+a) + (a+c)/((a+b)3)**

În acest caz, *E*, *a*, *b*, *c* şi *d* sunt identificatori pentru datele prelucrate de calculator. Data *E* este o dată de ieşire (rezultatul obţinut în urma evaluării expresiei), iar *a*, *b*, *c* şi *d* sunt date de intrare (date care se furnizează calculatorului pentru a putea calcula valoarea lui *E*).

Ordinea operaţiilor care se vor executa pentru a evalua expresia *e*:

e ← ((a/b+c)*4 + a) mod b > a*b or not (a+c)*b = a*c

care conţine paranteze şi mai multe tipuri de operatori, este:

e1 ← a/b (operator aritmetic de prioritate 2)
e2 ← e1 + c (operator aritmetic de prioritate 3)
e3 ← e2*4 (operator aritmetic de prioritate 2)
e4 ← e3 + a (operator aritmetic de prioritate 3)
e5 ← a + c (operator aritmetic de prioritate 3)
e6 ← e4 mod b (operator aritmetic de prioritate 2)
e7 ← a*b (operator aritmetic de prioritate 2)
e8 ← e5*b (operator aritmetic de prioritate 2)
e9 ← a*c (operator aritmetic de prioritate 2)
e10 ← e6 > e7 (operator relaţional)
e11 ← e8=e9 (operator relaţional)
e12 ← not e11 (operator logic)
e ← e10 or e11 (operator logic)

De exemplu, următoarea expresie matematică, prin care se calculează valoarea lui *e*:

$$\frac{\sqrt{a+1} - \sqrt{a-1}}{\sqrt{a+1} + \sqrt{a-1}} \times (2-a)$$

necesită calculul radicalului de ordin 2, pentru care se foloseşte funcţia **sqrt(*x*)**. Astfel, atunci când data *x* are valoarea 9, funcţia *sqrt(x)* furnizează valoarea 3, iar când *x* are valoarea 49, funcţia *sqrt(x)* furnizează valoarea 7. Pentru a putea fi evaluată de calculator, expresia va fi scrisă astfel:

e ← ((sqrt(a+1) - sqrt(a-1))/ (sqrt(a+1) + sqrt(a-1))) * (2 - a)

Ordinea operaţiilor care se execută pentru a evalua expresia *e* este:

e1 ← a+1	e5 ← sqrt(e1)	e9 ← e5-e6
e2 ← a-1	e6 ← sqrt(e2)	e10 ← e7+e8
e3 ← a+1	e7 ← sqrt(e3)	e11 ← e9/e10
e4 ← a+1	e8 ← sqrt(e4)	e12 ← 2 - a
		e ← e11 * e12

Precondiţiile unei expresii

Expresia defineşte cum trebuie să prelucreze calculatorul anumite valori ale datelor. De fiecare dată când calculatorul evaluează expresia, sunt prelucrate valorile curente ale datelor variabile care apar ca operanzi în cadrul expresiei. În unele cazuri trebuie stabilite anumite condiţii pentru valorile acestor date, pentru a se putea evalua expresia, adică trebuie stabilite **precondiţiile expresiei**.

> **Precondiţiile expresiei** reprezintă un ansamblu de restricţii şi constrângeri impuse datelor care apar în expresie ca operanzi.

Aceasta înseamnă că, dacă notăm precondiţia cu **P** şi expresia cu **E**, atunci:

- ✓ Dacă precondiţia **P** este adevărată (are valoarea *true*), expresia poate fi calculată cu acele valori ale datelor.
- ✓ Dacă precondiţia **P** este falsă (are valoarea *false*), expresia nu poate fi calculată cu acele valori ale datelor, deoarece fie se obţine o valoare eronată, fie se poate provoca o eroare de execuţie a programului.

Precondiţiile cele mai des întâlnite sunt:

- ✓ **Constrângerile operaţiei de împărţire**. În algoritmi puteţi folosi trei operatori pentru împărţire: /, **div** şi **mod**. Pentru toate aceste operaţii există **precondiţia ca împărţitorul să fie diferit de zero**. De exemplu, pentru expresia *a/(c+d)*, precondiţia este *c+d*<>0.
- ✓ **Constrângerile argumentului unei funcţii**. Argumentele unor funcţii trebuie să îndeplinească o anumită condiţie. De exemplu, funcţia ***sqrt(x)*** nu se poate aplica pe o valoare negativă (nu se poate extrage radicalul pătrat dintr-o valoare negativă). De exemplu, pentru expresia *-b+sqrt(b*b-4*a*c)* precondiţia este *b*b-4*a*c>=0*, iar pentru expresia *(-b+sqrt(b*b-4*a*c)/(2*a)* precondiţia este *(a<>0) and (b*b-4*a*c>=0)*.
- ✓ **Constrângerile pentru ca formula să fie validă**. Formula folosită pentru a calcula o valoare poate fi validă numai pentru anumite valori ale datelor care sunt folosite ca operanzi. De exemplu, operatorii **div** şi **mod** se pot aplica numai pe date de tip întreg.

Evaluare

Răspundeți:

1. Ce este identificatorul unei date? Dați cinci exemple de identificatori.
2. Ce este o variabilă de memorie?
3. Prin ce se deosebesc datele de intrare, datele de ieşire şi datele de manevră? Dați un exemplu de problemă în care să folosiți toate aceste tipuri de date. Precizați pentru fiecare dată de ce tip este.
4. Ce este tipul datei? Cum influențează tipul datei o variabilă de memorie?
5. Câte tipuri de date există? Ce operatori puteți folosi pentru fiecare tip de dată?
6. Ce este precedența operatorilor? Dar asociativitatea operatorilor?

Rezolvați:

1. Se consideră următoarea problemă:
   *Se dau două numere întregi **a** şi **b**. În funcție de răspunsul la un mesaj întrebare (de exemplu, "Ce operație doriți?") se va calcula: dacă răspunsul este litera x, media aritmetică a celor două numere; dacă răspunsul este litera y, media geometrică a celor două numere; dacă răspunsul este litera z, câtul şi restul împărțirii numărului a la numărul b; dacă răspunsul este orice altă literă, se va afişa un mesaj de informare (de exemplu, "Alegere greşită").*

Pentru rezolvarea acestei probleme cu ajutorul unui program de calculator, se vor folosi mai multe date elementare, care să permită generalizarea problemei. Completați următorul tabel, prin care veți face o analiză a datelor folosite:

Identificatorul datei	**Reprezintă**	**Tipul datei** (de intrare, de ieşire, ...)	**Tipul datei** (numeric, logic, ...)	**Observații** (constantă, formula de calcul, ...)

2. Legați prin linii fiecare element din coloana **Construcția** de elementul corespunzător din coloana **Reprezintă**:

Construcția:	
alfa	1
"alfa'	2
'10'''	3
'20''''	4
'alfa'	5
5000	6
"alfa"	7
'500'	8
"120"	9

Reprezintă:

a **identificator dată elementară**

b **constantă de tip şir de caractere**

c **constantă de tip numeric**

d **construcție greşită**

3. Dacă într-un algoritm există variabilele *a* de tip caracter, *b* de tip întreg şi *c* de tip logic, şi se atribuie acestor date următoarele valori inițiale,

   a: '4' **b:** 8 **c:** false

evaluaţi următoarele expresii:

Expresia	Rezultat	Expresia	Rezultat
(b>15) **or** c		**not** c **or** (a='a')	
a>='0' **and** a<='9'		(a > b) **and** c	

4. Considerând datele *x, y, z* de tip *real* şi *i, j, k* de tip *întreg*, specificaţi care dintre următoarele expresii sunt valide. Pentru expresiile valide, precizaţi tipul.

Expresia	Valid? (D/N)	Tip rezultat	Expresia	Valid? (D/N)	Tip rezultat
x+y+i			(i/j) **mod** k		
10(i+j)			2x-3y		
i **div** j			x*y-z*z/**sqrt**(i)		

5. Pentru următoarele valori ale datelor:

i [4] j [6] k [8] l [1] m [10]

evaluaţi următoarele expresii:

Expresia	Rezultat	Expresia	Rezultat
i+j*l-m		i-j+k-l+m	
(i+j) **div** (m-j)		3*j **mod** m -i	
(j+(m-l*(j+i))-k)+5		j*j-i-l	

6. Se dau următoarele secvenţe de operaţii de atribuire:

(S1): a ← 1; b ← 2; c ← 3; d ← 4
(S2): a ← b; b ← c; c ← d; d ← a
(S3): b ← c; c ← d; d ← a; a ← b
(S4): c ← d; d ← a; a ← b; b ← c

Să se precizeze valorile variabilelor de memorie *a, b, c* şi *d* obţinute în urma executării operaţiilor de atribuire, în ordinea:

a) (S1); (S2); (S3)
b) (S1); (S3); (S2)
c) (S1); (S2); (S2); (S3)
d) (S1); (S2); (S3); (S4)

Ce concluzii puteţi trage în urma executării operaţiilor de la punctele a) şi b)?

7. Scrieţi următoarele expresii matematice, în forma acceptată de calculator:

$$E_1 = \frac{x^3 - 1}{x(x-3)(x-1)}$$

$$E_2 = \frac{a^2 + (a+b)^3}{ab^2} + a^2 b$$

$$E_3 = \frac{4(\frac{a}{bc} + c) + (ab)^2 + 3\frac{a}{bc}}{2a^3}$$

8. Descrieţi ordinea de evaluare a celor trei expresii de la problema 7.

9. Scrieţi următoarea expresie matematică în forma acceptată de calculator folosind, pentru calculul radicalului de ordin 2 din x, funcţia **sqrt**(*x*):

$$\sqrt{\sqrt{\sqrt{x^2 - 1}}}$$

10. Descrieţi ordinea de evaluare şi calculaţi valoarea următoarei expresii, în funcţie de valorile care vor fi atribuite datelor *a* şi *b*:

e ← a and b or (not a and not b)

Pentru evaluarea expresiei veţi completa următorul tabel:

a	b	not a	not b	e1←not a and not b	e2← a and b	e← e2 or e1
T	*T*	*F*	*F*	*F*	*T*	*T*
T	*F*					
F	*T*					
F	*F*					

11. Descrieţi ordinea de evaluare şi calculaţi valoarea următoarei expresii, în funcţie de valorile care vor fi atribuite datelor *a*, *b* şi *c*. Pentru calcularea valorii expresiei, folosiţi modelul de tabel de la exerciţiul 10, care va avea opt linii în loc de patru, corespunzătoare tuturor combinaţiilor posibile pentru cele trei date.

e ← (*not* a *or* b) *and* ((*not* a *and not* c) *or* (a *or* b *or* c))

12. Ce valori poate lua următoarea expresie: **e ← b-4>1 *and* a+b> 0**, dacă operanzii pot lua următoarele valori: *a* ∈ {1, -10, -20} şi *b* ∈ {4, 16}. Câte date se folosesc pentru evaluarea acestei expresii şi de ce tip sunt? Precizaţi care sunt datele de intrare şi care sunt cele de ieşire.

13. Pentru fiecare dintre următoarele secvenţe de operaţii de atribuire, scrieţi o singură operaţie de atribuire, astfel încât să obţineţi valoarea atribuită ultimei variabile din secvenţă:

Secvenţa a	Secvenţa b
a ← a+1 c ← 2 d ← b+a e ← 4*c-d	x ← **sqrt**(a+b) y ← a-b z ← 20*x-y
e ←	z ←

14. Alegeţi mulţimea în care poate lua valori expresia *e*, pentru a ∈ {-5, -10, -20} şi b ∈ {4, 16}; ***e ← b+4>1 OR a+b<0***.

a) {*T*,*F*} b) {*T*} c) {*F*}

adică este o dată: a) constantă; b) variabilă.

15. Scrieţi precondiţiile următoarelor expresii:

a) a/b b) **sqrt**(a+1) c) (x+y) **mod** z d) **sqrt**(a*a+b*b)

16. În unele aplicaţii este necesar să se precizeze că o dată poate lua numai anumite valori. Aceste descrieri se folosesc pentru a verifica dacă valoarea datei obţinută în urma unor operaţii de prelucrare aparţine mulţimii de valori permise. De exemplu, considerăm mulţimea A=(0,10]∪{20} şi o dată de tip numeric *x*. Pentru a verifica dacă *x*∈A, se construieşte o expresie logică *c1* care descrie condiţia de apartenenţă a datei *x* la mulţimea A, urmând ca prin testarea valorii acestei expresii să se verifice dacă valoarea datei *x* este corectă: dacă expresia *c1* are valoarea adevărat, *x*∈A, iar dacă expresia *c1* are valoarea fals, *x*∉A. Pentru acest exemplu se construieşte expresia *c1*:

c1 ← (x>0 *and* x<=10) *or* x = 20

Dacă vreţi să verificaţi că valoarea datei *x* nu trebuie să aparţină mulţimii precizate mai sus, se va construi expresia *c2*, prin negarea expresiei *c1*:

c2 ← *not* c1 = *not* ((x>0 *and* x<=10) *or* x = 20) = *not* (x>0 *and* x<=10) *and not* (x = 20) =
= (*not* x>0 *or not* x<=10) *and* x<>20 = (x<=0 *or* x>10) *and* x<>20

Observați că, în urma negării, un operator relațional se înlocuiește cu complementarul său: <> cu =, >= cu <, <= cu >, și invers.

Pornind de la acest exemplu, să considerăm mulțimea $A=[-10,10]\cup(20,30)\cup\{50\}$ și o dată de tip numeric *x*. Precizați expresia *c1* prin care se verifică dacă $x\in A$ și expresia *c2* prin care se verifică dacă $x\notin A$.

17. Fie mulțimea $A = [-30,-20] \cup (-15, 10] \cup (40, 50) \cup \{-50, 100\}$ și o dată de tip numeric *x*. Precizați expresia *c1* prin care se verifică dacă $x\in A$ și expresia *c2* prin care se verifică dacă $x\notin A$.

18. De exemplu, dacă datele *a*, *b* și *c* reprezintă lungimile laturilor unui triunghi ABC, pentru a verifica dacă acest triunghi este un triunghi echilateral (adică: *a=b=c*) trebuie să se evalueze expresia logică *c1*:

c1 ← (a = b) *and* (b = c)

Dacă expresia are valoarea „adevărat", triunghiul este echilateral. Această expresie descrie condiția pe care trebuie să o îndeplinească cele trei date *a*, *b* și *c* ca să reprezinte laturile unui triunghi echilateral. Pornind de la acest exemplu, precizați condiția pe care trebuie să o îndeplinească cele trei date pentru ca triunghiul să fie:

a) isoscel, b) dreptunghic, c) dreptunghic isoscel.

19. Precizați **condiția** pe care trebuie să o îndeplinească trei date, ca să reprezinte lungimile laturilor unui triunghi (suma lungimilor a două laturi trebuie să fie mai mare decât lungimea celei de a treia laturi, oricare ar fi cele două laturi).

20. Scrieți condiția *c1* prin care testați dacă patru date de tip numeric *a*, *b*, *c* și *d* pot reprezenta laturile unui paralelogram, și condiția *c2*, prin care testați dacă aceste date nu pot reprezenta laturile unui paralelogram.

21. Scrieți condiția prin care testați dacă valoarea unui număr întreg *n* este:

a) un număr impar,
b) un număr divizibil cu 3 sau cu 5,
c) un număr divizibil cu 3 și cu 5,
d) un număr divizibil cu 3 dar nu și cu 5,
e) un pătrat perfect (veți folosi funcțiile: **int**(x) – pentru partea întreagă din *x* și **sqrt**(x) – pentru radical de ordinul 2 din *x*).

22. Pentru a testa ultima cifră a unui număr întreg *n*, aceasta se extrage cu expresia ***cifra ← n mod 10*** (restul împărțirii numărului la 10). De exemplu, pentru a testa dacă ultima cifră a unui număr *n* este 2, se folosește condiția ***c ← n mod 10 = 2***. Scrieți condiția prin care testați dacă ultima cifră a unui număr *n* este:

a) 3 sau 5, b) diferită de 3 și de 5, c) impară, d) pară,
e) multiplu de 3, f) multiplu de 3 sau de 5.

23. În data *alfa* se memorează un caracter. Pentru a afla dacă acest caracter este litera *a* sau *A*, se va testa expresia *c1* care descrie această **condiție**:

c1 ← (alfa = "a") *or* (alfa = "A")

iar pentru a afla dacă acest caracter este una dintre literele *a*, *b*, *c*, *d*, *A*, *B*, *C* sau *D*, se va testa expresia *c2* care descrie această condiție:

c2 ← (alfa >= "a" *and* alfa <= "d") *or* (alfa >= "A" *and* alfa <= "D")

Pornind de la acest exemplu precizaţi condiţia prin care se poate testa caracterul memorat în data *alfa* astfel încât:

a) să nu fie o cifră (de exemplu "1" sau "8"),
b) să fie o vocală,
c) să aparţină mulţimii de caractere C= {*a, x, y, z, w, B, X, Y, Z*},

24. Se consideră următorul enunţ: *Se citesc n numere naturale. Să se calculeze suma şi produsul celor divizibile cu 3 sau cu 5.* Analizaţi problema şi scrieţi algoritmul.

25. Se consideră următorul enunţ: *Se citesc n numere naturale. Să se numere câte sunt divizibile cu 2, cu 3 şi, respectv, cu 6.* Analizaţi problema şi scrieţi algoritmul.

26. Se consideră următorul enunţ: *Se citesc mai multe numere naturale până la întâlnirea numărului 0. Să se calculeze media aritmetică a numerelor care au ultima cifră 5.* Analizaţi problema. Identificaţi cazul în care trebuie să folosiţi precondiţia expresiei. Scrieţi algoritmul.

Alegeţi:

1. Datei *a* i se atribuie valoarea "2+1=". Dacă se afişează conţinutul ei pe ecran, veţi vedea:

a) "3" b) 3 c) "2+1=" d) 2+1=

2. Datei *a* i se atribuie expresia "25"+"75". Valoarea datei va fi:

a) "2575" b) 100 c) "100"

3. Prin negarea expresiei ***n mod 2<>0 OR n>10*** se testează dacă data *n* este:

a) un număr par mai mare decât 10;
b) un număr impar mai mare decât 10;
c) un număr par mai mic sau egal cu 10;
d) un număr impar mai mic sau egal cu 10;

4. Care dintre expresiile următoare testează dacă *n* este un număr natural multiplu de 3 sau de 5:

a) n>0 and n mod 3 = 0 or n mod 5 = 0
b) n>0 and n mod 3 = 0 and n mod 5 = 0
c) n>0 and (n mod 3 = 0 or n mod 5 = 0)
d) n>0 or n mod 3 = 0 or n mod 5 = 0

5. Care dintre expresiile următoare testează dacă *n* este un număr natural care nu se divide prin 3 şi prin 5:

a) n>0 and n mod 3 <> 0 or n mod 5 <> 0
b) n>0 and n mod 3 <> 0 and n mod 5 <> 0
c) n>0 and (n mod 3 <> 0 or n mod 5 <> 0)
d) n>0 or n mod 3 <> 0 or n mod 5 <> 0

6. Care dintre expresiile următoare testează dacă *n* este un număr natural care are ultima cifră diferită de 5 şi de 0:

a) n>0 and n mod 10 <> 5 or n mod 10 <> 0
b) n>0 and n mod 10 <> 5 and n mod 10 <> 0
c) n>0 and (n mod 10 <> 5 or n mod 10 <> 0)
d) n>0 or n mod 10 <> 5 or n mod 10 <> 0

7. Care dintre expresiile următoare testează dacă *n* este un număr natural care are ultima cifră 0, 2, 4 sau 8:

a) n>0 and n mod 10 mod 2 = 0 or n mod 10 mod 3 <> 0
b) n>0 and n mod 10 mod 2 = 0 and n mod 10 mod 3 <> 0
c) n>0 and (n mod 10) mod 2 = 0 or (n mod 10) mod 3 <> 0
d) n>0 and ((n mod 10) mod 2 = 0 or (n mod 10) mod 3 <> 0)

8. În urma cărei operații de atribuire, data *b* va avea valoarea ultimei cifre a numărului întreg *a*:

a) b ← a div 10
b) b ← a-a div 10*10
c) b ← a-a div 10
d) b ← a-a mod 10

3. Algoritmii

3.1. Reprezentarea algoritmilor

Algoritmul este un concept abstract. **Reprezentarea algoritmului înseamnă implementarea fizică a algoritmului**. Chiar dacă algoritmul este unic, el poate avea mai multe reprezentări fizice. De exemplu, algoritmul de rezolvare a ecuației de gradul întâi poate fi reprezentat prin calculele efectuate pe hârtie de fiecare dată când se rezolvă manual o ecuație de gradul întâi, prin circuite electronice, dacă s-ar construi o maşină electronică numai pentru rezolvarea ecuației de gradul întâi, sau prin instrucțiunile unui program care descriu pentru calculator algoritmul de rezolvare.

Când construiți un algoritm trebuie să țineți cont de următoarele reguli:

- ✓ să definiți exact datele asupra cărora lucrează algoritmul (datele de intrare, datele de ieşire şi datele intermediare);
- ✓ să definiți exact operațiile care se vor executa cu datele asupra cărora lucrează algoritmul;
- ✓ să definiți exact noțiunea de structură de control a algoritmului;
- ✓ să definiți exact succesiunea de structuri care formează algoritmul.

Algoritmul prin care se descrie o problemă care trebuie să fie rezolvată de calculator nu trebuie să fie ambiguu deoarece, în cazul exprimărilor neclare, calculatorul nu poate să opteze singur pentru o anumită posibilitate. Pentru a evita ambiguitatea descrierii unui algoritm printr-un limbaj natural (limba în care vorbim), se poate folosi pentru reprezentarea lui un limbaj artificial numit **pseudocod**, apropiat de limbajul de programare, dar care este puțin formalizat şi nu este constrâns de regulile de sintaxă ale limbajului de programare (de exemplu, în pseudocod se poate folosi exprimarea **dacă...atunci...altfel** – în limba română – sau formularea **if...then...else** – în limba engleză – care sunt foarte aproapiate de limbajul natural, dar care permit descrierea unor operații specifice din algoritm).

Pseudocodul (codul fals) este considerat un cod fals deoarece nu poate fi folosit pentru a exprima instrucțiunile care se dau calculatorului pentru a rezolva problema descrisă de algoritm (nu poate fi folosit ca limbaj de programare). El foloseşte expresii, din limbajul natural, în care exprimarea acțiunilor care se execută se face prin propoziții care se termină prin simbolul punct şi virgulă (;). În propoziții se folosesc **cuvinte cheie**, pentru descrierea structurilor de control şi a operațiilor de comunicare. O propoziție care reprezintă un pas de comunicare sau de acțiune începe obligatoriu cu un verb.

Pseudocodul permite şi descrierea datelor asupra cărora acționează algoritmul. Pentru precizarea tipului de dată se folosesc cuvinte cheie. De exemplu, se pot folosi următoarele cuvinte cheie: **întreg** – tipul numeric întreg; **real** – tipul numeric real; **logic** – tipul logic; **caracter** – tipul caracter; şi **şir** – tipul şir caracter. Cuvântul cheie care precizează tipul este urmat de o listă prin care se enumeră identificatorii datelor care corespund acelui tip.

În cazul pasului de comunicare, verbul este **citeşte** (**read**) pentru o operaţie de intrare şi **scrie** (**write**) pentru o operaţie de ieşire. Verbul este urmat de lista datelor care se comunică. Lista conţine date variabile reprezentate prin identificatorii lor. În cazul unei operaţii de scriere lista poate conţine şi date constante reprezentate prin valoare (de exemplu, un mesaj reprezentat printr-o constantă de tip şir de caractere). Elementele listei se separă prin virgulă.

Cuvintele cheie pot fi în limba română (pseudocodul în limba română) sau în limba engleză (pseudocodul în limba engleză). Pseudocodul în limba engleză este mai aproape de cuvintele cheie folosite în instrucţiunile unui limbaj de programare. Din această cauză vor fi prezentate ambele versiuni de pseudocod, urmând ca în aplicaţii să folosim numai pseudocodul în limba română.

Pentru a delimita secvenţa de descriere a datelor de secvenţa de descriere a paşilor algoritmului, paşii algoritmului vor fi încadraţi de cuvintele cheie **început... sfârşit.** (**begin... end.**).

De exemplu, pseudocodul pentru descrierea algoritmului de rezolvare a ecuaţiei de gradul întâi este:

```
real a,b,z;
început
 citeşte a,b;
 dacă a=0
   atunci
     dacă b=0
       atunci scrie "Ecuaţia are o infinitate de soluţii";
       altfel scrie "Ecuaţia nu are soluţii";
     sfârşit_dacă;
   altfel z←-b/a;
          scrie "Soluţia ecuaţiei este ", z;
 sfârşit_dacă;
sfârşit.
```

Observaţii:

1. În pseudocod, operaţia de comparaţie a fost exprimată prin cuvintele cheie **dacă** şi **sfârşit_dacă**. Aceste două cuvinte formează o **structură de control**, adică o entitate din cadrul algoritmului prin care se descrie modul în care paşii algoritmului îşi predau controlul unul altuia.
2. În pseudocod, pentru orice structură de control se foloseşte o **pereche de cuvinte cheie**: primul cuvânt precizează începutul structurii (cuvântul **dacă**), iar al doilea cuvânt precizează sfârşitul structurii (cuvântul **sfârşit_dacă**).
3. Ca să fiţi siguri că aţi scris corect structurile de control, verificaţi ca numărul de structuri deschise să fie egal cu numărul de structuri închise (de exemplu, numărul de cuvinte **dacă** să fie egal cu numărul de cuvinte **sfârşit_dacă**).
4. Pentru a urmări mai uşor dacă structurile sunt închise corect, scrieţi indentat corpul structurii faţă de cuvintele cheie cu care începeţi şi cu care terminaţi structura. Scrieţi la acelaşi nivel de indentare propoziţiile care se execută secvenţial. De exemplu, în structura de control descrisă în pseudocod prin cuvintele

cheie **dacă... sfârşit_dacă**, scrieţi indentat faţă de cuvintele cheie **atunci** şi **altfel** paşii care se execută pentru cele două valori ale condiţiei testate.

3.2. Principiile programării structurate

Algoritmul este format din paşii care urmează să se execute şi ordinea în care se vor executa pentru a rezolva problema. Algoritmul pentru rezolvarea ecuaţiei de gradul întâi conţine nouă paşi (vezi algoritmul de la pagina 9), iar ordinea de executare depinde de valoarea celor doi coeficienţi **a** şi **b**:

✓ Dacă **a** =0 şi **b** =0, se execută în ordine paşii: ***1, 2, 3, 4, 5, 9.***
✓ Dacă **a** =0 şi **b** ≠0, se execută în ordine paşii: ***1, 2, 3, 4, 6, 9.***
✓ Dacă **a** ≠ 0, se execută în ordine paşii: ***1, 2, 3, 7, 8, 9.***

Structura de control a algoritmului defineşte **ordinea de executare a paşilor**, adică ordinea în care un pas predă controlul altui pas şi prin care se determină fluxul controlului. În cadrul algoritmilor pot fi folosite trei tipuri de structuri de control:

- **structura liniară**
- **structura alternativă**
- **structura repetitivă**

3.2.1. Structura liniară

Structura liniară sau secvenţială este structura în care paşii se execută în ordinea în care au fost scrişi: ***Pasul 1**, **Pasul 2**, ..., **Pasul i**, **Pasul i+1**, ..., **Pasul n-1**, **Pasul n**.* Fiecare pas predă controlul pasului următor (***Pasul i*** predă controlul ***Pasului i+1***). Un pas al structurii secvenţiale se execută numai dacă au fost executaţi toţi paşii care îl preced. Structura secvenţială nu foloseşte decât paşi de tip acţiune şi comunicare (pentru datele de intrare şi datele de ieşire).

Exemplu

Se introduc de la tastatură trei numere **a**, **b**, **c**. Să se calculeze media aritmetică dintre **a** şi **b** şi media aritmetică dintre **b** şi **c**. Pentru calcularea celor două medii se vor folosi două variabile de memorie, **m1** şi **m2**, şi se vor executa, în ordine, următoarele acţiuni:

Pasul 1. *Început.*
Pasul 2. ***Comunică** valorile pentru **a, b** şi **c**.*
Pasul 3. ***Calculează m1 ← (a+b)/2.***
Pasul 4. ***Calculează m2 ← (b+c)/2.***
Pasul 5. ***Comunică** valorile lui **m1** şi **m2**.*
Pasul 6. *Terminat.*

```
întreg a,b,c;
real m1,m2;
început
 citeşte a,b,c;
 m1 ← (a+b)/2;
 m2 ← (b+c)/2;
 scrie m1,m2;
sfârşit.
```

Ordinea în care se execută cei şase paşi este ordinea prezentată: ***1, 2, 3, 4, 5, 6,*** oricare ar fi valorile pentru **a**, **b**, **c**. ***Pasul 1*** predă controlul ***Pasului 2***, ***Pasul 2*** predă controlul ***Pasului 3***, ***Pasul 3*** predă controlul ***Pasului 4***, ***Pasul 4*** predă controlul ***Pasului 5*** şi ***Pasul 5*** predă controlul ***Pasului 6***.

Algoritmul a fost transpus în pseudocod.

Probleme care se pot rezolva cu ajutorul structurii secvenţiale:

1. Se citesc trei numere întregi, de la tastatură. Să se calculeze media aritmetică.
2. Se citesc dimensiunile pentru laturile unui triunghi. Să se calculeze aria şi perimetrul triunghiului.

3.2.2. Structura alternativă

Prin această structură, se face selectarea între două sau mai multe acţiuni, în funcţie de anumite condiţii.

Există două tipuri de structuri alternative:

- **structura alternativă simplă,**
- **structura alternativă generalizată.**

Structura alternativă simplă

La acest tip de structură se face selectarea între două acţiuni, în funcţie de îndeplinirea sau neîndeplinirea unei condiţii. De exemplu:

- ✓ Dacă anul este bisect, atunci împarte totalul la 366; altfel, împarte la 365. Condiţia este tipul anului: bisect sau nu.
- ✓ Dacă vânzările au scăzut, reduceţi preţul cu 5%. Condiţia este scăderea vânzărilor.
- ✓ Dacă unghiul are 90^0, este un unghi drept. Condiţia este valoarea unghiului.
- ✓ Dacă triunghiul are toate laturile egale, este echilateral. Condiţia este relaţia de egalitate între laturile triunghiului.
- ✓ Dacă patrulaterul are toate laturile egale şi un unghi de 90^0, este un pătrat. Condiţia este relaţia de egalitate între laturile patrulaterului şi valoarea unui unghi al patrulaterului.
- ✓ Dacă ai sub 18 ani, atunci eşti minor; sau dacă ai peste 18 ani, atunci eşti major. Condiţia este vârsta de 18 ani.
- ✓ Dacă vei fi în oraş la ora 13, luăm prânzul împreună. Condiţia este prezenţa în oraş la ora 13.
- ✓ Dacă nu ai 10.000.000 de lei, împrumută-te la bancă. Condiţia este suma de 10.000.000 lei.
- ✓ Dacă valoarea polinomului este 0 pentru x=2, polinomul se divide prin x-2. Condiţia este valoarea 0 a polinomului pentru x=2.

Exemplu

Se introduce de la tastatură un număr **n**. Să se calculeze inversul acestui număr, **inv**, definit astfel:

$$inv=\begin{cases} 1/n \text{ pentru } n\neq 0 \\ 0 \text{ pentru } n=0 \end{cases}$$

Altfel spus: dacă n este diferit de 0, inversul are valoarea 1/n; altfel, are valoarea 0.

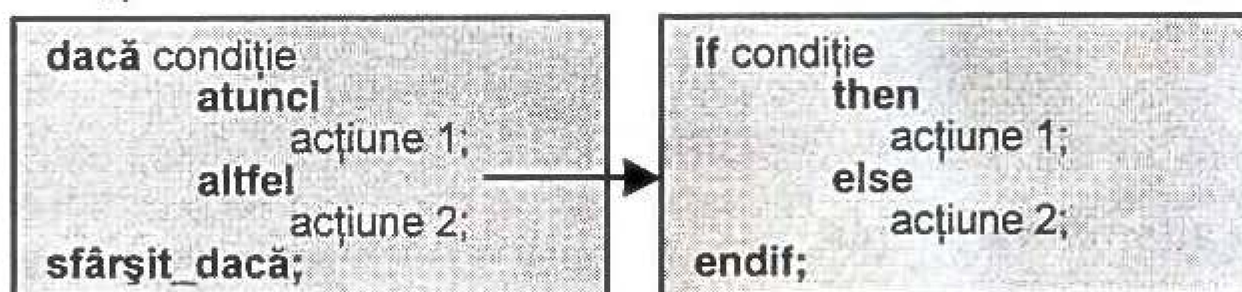
Pentru calculul inversului se vor executa, în ordine, următoarele acţiuni:

Pasul 1. *Început.*
Pasul 2. ***Comunică** valoarea pentru **n**.*
Pasul 3. ***Dacă** n<>0, **atunci*** execută ***Pasul 4**, **altfel*** execută ***Pasul 5**.*
Pasul 4. ***Calculează inv ← 1/n.***
Pasul 5. ***Calculează inv ← 0.***

Pasul 6. ***Comunică*** *valoarea lui* ***inv.***
Pasul 7. *Terminat.*

Executarea uneia dintre cele două acțiuni posibile depinde de condiția precizată printr-o expresie logică. Dacă expresia logică are valoarea „adevărat", se execută *acțiune 1*, iar dacă expresia logică are valoarea „fals", se execută *acțiune 2*. Cele două acțiuni pot fi descrise printr-un singur pas sau prin mai mulți pași.

Observați că acest tip de structură poate fi descris prin raționamentul **dacă... atunci... altfel... sfârşit_dacă** (**if... then... else... endif**) pe care îl putem folosi în pseudocod, pentru descrierea structurii.



Cuvântul **dacă** (**if**) marchează începutul structurii, iar cuvântul **sfârşit_dacă** (**endif**) sfârşitul structurii. Cuvintele **dacă** (**if**) şi **atunci** (**then**) delimitează expresia logică ce se evaluează, cuvântul **atunci** (**then**) marchează începutul secvenței de paşi care descriu acțiunea care se va executa dacă expresia logică are valoarea *adevărat*, şi cuvântul **altfel** (**else**) marchează începutul secvenței de paşi care descriu acțiunea care se va executa dacă expresia logică are valoarea *fals*.

```
întreg n;
real inv;
început
  citeşte n;
  dacă n<>0
     atunci
        inv←1/n;
     altfel
        inv←0;
  sfârşit_dacă;
  scrie inv;
sfârşit.
```

Observați că se execută secvențial ***Pasul 1***, ***Pasul 2***, ***Pasul 3***, după care ordinea secvențială este abandonată şi se execută ***Pasul 4*** sau ***Pasul 5***, în funcție de valoarea expresiei logice ***n<>0***, după care se reia ordinea secvențială, executându-se ***Pasul 6*** şi ***Pasul 7***.

Exemplul a fost transpus în pseudocod.

Un caz particular de structură alternativă este **structura alternativă cu o ramură vidă**, în care *acțiune 2* nu conține nici un pas. Deoarece pentru valoarea *fals* a expresiei logice nu se execută nici o acțiune, din pseudocod va fi eliminat cuvântul **altfel** (**else**) care marchează începutul secvenței de paşi ce descriu acțiunea care se va executa dacă expresia logică are valoarea *fals*.

Exemplu

Se introduce de la tastatură un număr **n**. Să se înlocuiască numărul **n** cu modulul său, definit astfel:

$$\mathbf{mod(n)}=\begin{cases} \mathbf{n} \text{ pentru } \mathbf{n} \geq \mathbf{0} \\ -\mathbf{n} \text{ pentru } \mathbf{n} < \mathbf{0} \end{cases}$$

Se vor executa, în ordine, următoarele acțiuni:

Pasul 1. *Început.*
Pasul 2. ***Comunică*** *valoarea pentru* ***n.***
Pasul 3. ***Dacă n<0, atunci*** *execută* ***Pasul 4, altfel*** *execută* ***Pasul 5.***
Pasul 4. ***Calculează n ← -n.***

Pasul 5. ***Comunică*** *valoarea lui* ***n***.
Pasul 6. *Terminat.*

Exemplul a fost transpus în pseudocod.

```
întreg n;
început
  citeşte n;
  dacă n<0
   atunci
      n ←-n;
  sfârşit_dacă;
  scrie n;
sfârşit.
```

Observaţie: În cazul structurii alternative cu o ramură vidă, acţiunea se va executa pe ramura pentru care condiţia are valoarea adevărat. Dacă, după ce aţi gândit algoritmul, acţiunea se execută pe ramura pentru care condiţia are valoarea *fals*, îl veţi corecta prin negarea condiţiei.

Structura alternativă generalizată

La acest tip de structură, se face selectarea între mai multe acţiuni, în funcţie de o variabilă de memorie numită **selector**, care poate lua mai multe valori, dintr-o mulţime ordonată de elemente de acelaşi tip cu selectorul. De exemplu:

✓ Dacă simbolul dintre doi operanzi este **+**, adună operanzii, dacă este minus, scade din primul operand al doilea operand, dacă este *****, înmulţeşte operanzii, iar dacă este **/**, împarte operanzii, iar dacă este alt operand, expresia este eronată. În acest caz selectorul este simbolul, iar mulţimea de valori este formată din simbolurile: **+**, **-**,***** şi **/**.

✓ Dacă este clasa a IX-a A, sunt 27 de elevi, dacă este clasa a IX-a B, sunt 28 de elevi, iar dacă este clasa a IX-a C, sunt 28 de elevi. În acest caz, selectorul este numele clasei, iar mulţimea de valori este formată din denumirerile de clase: a IX-a A, a IX-a B şi a IX-a C.

Exemplu

Se introduc de la tastatură numerele întregi **n**, **a**, **b** şi **c**. Să se calculeze valoarea lui **e**, definit astfel:

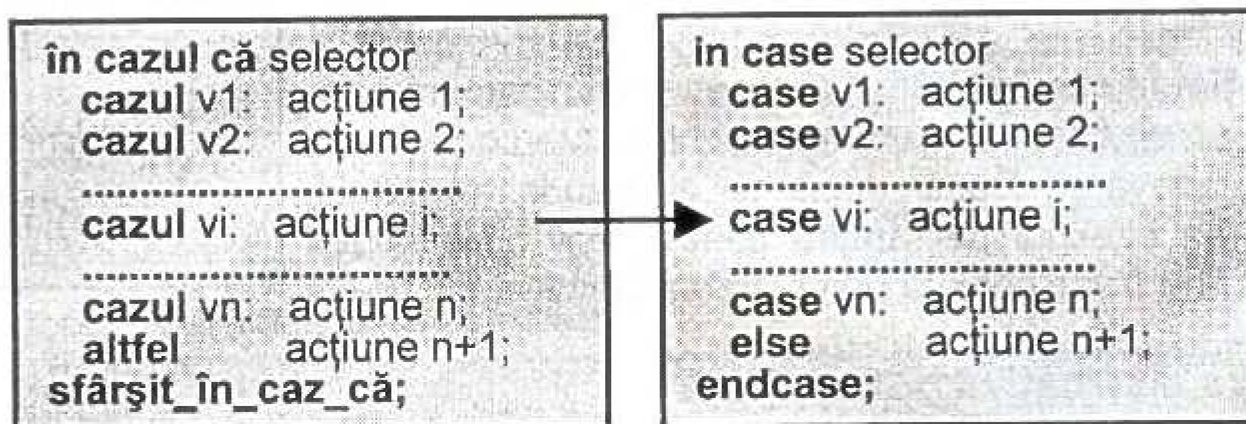
$$e = \begin{cases} (a+b)/c & \text{pentru } n=1 \\ (b+c)/a & \text{pentru } n=2 \\ (c+a)/b & \text{pentru } n=3 \end{cases}$$

Pentru calculul expresiei **e** se vor executa, în ordine, următoarele acţiuni:

Pasul 1. *Început.*
Pasul 2. ***Comunică*** *valorile pentru* ***n, a, b, c***.
Pasul 3. ***Dacă n=1, atunci*** execută ***Pasul 4, altfel*** execută ***Pasul 5.***
Pasul 4. ***Calculează e*** ← ***(a+b)/c***. *Treci la* ***Pasul 9.***
Pasul 5. ***Dacă n=2, atunci*** execută ***Pasul 6, altfel*** execută ***Pasul 7.***
Pasul 6. ***Calculează e*** ← ***(b+c)/a***. *Treci la* ***Pasul 9.***
Pasul 7. ***Dacă n=3, atunci*** execută ***Pasul 4, altfel*** execută ***Pasul 9.***
Pasul 8. ***Calculează e*** ← ***(c+a)/b***. *Treci la* ***Pasul 9.***
Pasul 9. ***Comunică*** *valoarea lui* ***e***.
Pasul 10. *Terminat.*

Executarea uneia dintre acţiunile posibile depinde de valoarea selectorului. Dacă selectorul are valoarea *v1*, se execută *acţiune 1*, dacă are valoarea *v2*, se execută *acţiune 2*, ..., dacă are valoarea *vn*, se execută *acţiune n*; altfel, se execută *acţiune n+1*. Fiecare valoare a selectorului corespunde unui **caz** tratat, căruia îi corespunde o acţiune care poate fi descrisă printr-un singur pas sau prin mai mulţi paşi.

Observați că acest tip de structură poate fi descris prin raționamentul **în cazul... caz1... caz2... altfel... sfârşit_în_cazul_că** (**in case... case1... case2... else... endcase**) pe care îl putem folosi în pseudocod, pentru descrierea structurii.


```
în cazul că selector
   cazul v1:  acţiune 1;
   cazul v2:  acţiune 2;
   ............................
   cazul vi:  acţiune i;
   ............................
   cazul vn:  acţiune n;
   altfel     acţiune n+1;
sfârşit_în_caz_că;
```

```
in case selector
   case v1:  acţiune 1;
   case v2:  acţiune 2;
   ............................
   case vi:  acţiune i;
   ............................
   case vn:  acţiune n;
   else      acţiune n+1;
endcase;
```

Cuvintele **în cazul că** (**in case**) marchează începutul structurii, iar cuvântul **sfârşit_în_caz_că** (**endcase**) sfârşitul structurii. Cuvântul **cazul** (**case**) marchează începutul secvenței de paşi care descriu acțiunea care se va executa pentru acel caz şi cuvântul **altfel** (**else**) marchează începutul secvenței de paşi care descriu acțiunea care se va executa dacă selectorul nu a avut valoarea nici unui caz.

```
întreg n,a,b,c;
real e;
început
  citeşte n,a,b,c;
  dacă n=1
     atunci e←(a+b)/c;
     altfel
       dacă n=2
         atunci e←(b+c)/a;
         altfel dacă n=3
                  atunci e←(c+a)/b;
                sfârşit_dacă;
       sfârşit_dacă;
  sfârşit_dacă;
  scrie e;
sfârşit.
```

Observație: Structura alternativă generalizată corespunde unor **structuri alternative simple imbricate**. O structură alternativă generalizată care are **n** cazuri este echivalentă cu **n** structuri alternative simple imbricate, în care condiția logică este dată de valoarea selectorului pentru acel caz.

Exemplul a fost transpus în pseudocod.

```
întreg n,a,b,c;
real e;
început
  citeşte n,a,b,c;
  în cazul că n
    caz 1:e←(a+b)/c;
    caz 2:e←(b+c)/a;
    caz 3:e←(c+a)/b;
  sfârşit_în_caz_că;
  scrie e;
sfârşit.
```

Probleme care se pot rezolva cu ajutorul structurii alternative:

1. Se citesc patru numere întregi de la tastatură: **a**, **b**, **c** şi **d**. Determinați care dintre produsele **a** × **b** şi **c** × **d** este mai mare.
2. Se citesc trei numere **a**, **b** şi **c**. Să se numere câte sunt pare.
3. Se citesc trei numere **a**, **b** şi **c**. Să se verifice dacă ele pot fi termenii unei progresii aritmetice.
4. Se citesc trei numere **a**, **b** şi **c**. Aflați dacă aceste numere pot reprezenta laturile unui triunghi. În caz afirmativ, calculați aria şi afişați ce tip de triunghi este (oarecare, isoscel, echilateral, dreptunghic sau dreptunghic isoscel).
5. Se citeşte un număr întreg **n** care reprezintă un an calendaristic. Să se verifice dacă anul este bisect sau nu (condiția ca un an să fie bisect este ca, dacă anul nu este divizibil cu 100, să fie divizibil cu 4; altfel, să fie divizibil cu 400).

6. Se citesc două intervale de timp exprimate în ore, minute şi secunde (**h1**, **m1** şi **s1**, pentru primul interval, şi **h2**, **m2** şi **s2**, pentru al doilea interval). Să se calculeze suma celor două intervale de timp.
7. Se citesc trei numere reale. Să se calculeze minimul şi maximul modulelor lor.

3.2.3. Structura repetitivă

Prin această structură se execută repetat o acţiune, sau o secvenţă de acţiuni, atât timp cât condiţia precizată este adevărată:

- ✓ Cât timp mai sunt bilete, vindeţi bilete, sau vindeţi bilete până le terminaţi.
- ✓ Cât timp semaforul este verde, mai trece o maşină.
- ✓ Cât timp mai aveţi numere, le adunaţi la sumă.
- ✓ Cât timp mai aveţi greşeli de corectat, corectaţi greşeli, sau corectaţi greşeli până aţi corectat şi ultima greşeală.
- ✓ Cât timp mai aveţi monede în buzunar, scoateţi o monedă sau scoateţi câte o monedă din buzunar până când nu mai aveţi nici o monedă.
- ✓ Începând de la numărul 1, scrieţi în ordine numerele până la 100.
- ✓ Începând de la numărul 2, scrieţi în ordine numerele pare până la 50.

Executarea repetată a unei acţiuni sau a unei secvenţe de acţiuni este un concept algoritmic foarte important. Metoda de implementare a unei astfel de repetiţii este **structura repetitivă** sau **iterativă**, cunoscută sub numele de **ciclu** sau **buclă** (*loop*), în care un grup de acţiuni, numit **corpul ciclului** sau **iteraţie**, se execută repetat, sub un **proces de control** (testează *condiţie*):

testează condiţie
execută corpul ciclului
testează condiţie
execută corpul ciclului
....
testează condiţie până când condiţia nu mai este îndeplinită.

Exemplul 1

Se introduc de la tastatură mai multe numere, până când ultimul număr este 0, şi trebuie să se calculeze suma numerelor. Se vor folosi două variabile de memorie: **s** (suma), care va avea iniţial valoarea 0, şi **a** (valoarea care se citeşte de la tastatură). Valoarea citită **a** se va aduna la sumă până când valoarea lui **a** va fi 0, adică putem spune: **cât timp** ***a<>0****, adună-l pe* **a** *la* s. Algoritmul va fi:

execută s ← 0
citeşte valoarea lui a
testează a <> 0
execută s ← s+a; citeşte a;
testează a<>0
execută s ← s+a; citeşte a;
....
testează a<>0 până când a=0 (condiţia nu mai este îndeplinită)
scrie valoarea lui s

Exemplul 2

Să se calculeze suma a **n** numere introduse de la tastatură. Se vor folosi patru variabile de memorie: **n** (câte numere se citesc), **i** (un contor care numără câte numere s-au citit), care va avea inițial valoarea 1 (urmează să fie citit primul număr), **s** (suma), care va avea inițial valoarea 0, și **a** (valoarea care se citește de la tastatură). Valoarea citită **a** se va aduna la sumă până când valoarea contorului **i** va fi mai mare decât **n**, adică putem spune: **pentru** *i<=n, adună-l pe* **a** *la* **s**. Algoritmul va fi:

citește n; execută s ← 0; execută i ← 1
testează i<=n
citește a; execută s ← s+a; execută i ← i+1
testează i<=n
citește a; execută s ← s+a; execută i ← i+1
....
testează i<=n până când i>n (condiția nu mai este îndeplinită)
scrie valoarea lui s

Procesul de control cuprinde trei acțiuni:

Inițializarea. Stabilește starea inițială, starea dinainte de prima parcurgere a corpului ciclului. În primul exemplu, inițializarea cuprinde operația de atribuire *s ← 0* și citirea primului număr (*citește a*). În al doilea exemplu, inițializarea cuprinde operațiile de atribuire *s ← 0* și *i ← 1*.

Testarea. Compară starea curentă cu starea care termină procesul de repetare și are rolul de a termina procesul de ciclare. Dacă cele două stări sunt egale, procesul de executare repetată a corpului ciclului se termină. În primul exemplu se compară valoarea numărului citit de la tastatură, **a**, cu **0** (*a<>0*), și se continuă executarea, repetată atât timp cât operatorul relațional furnizează valoarea „adevărat". Procesul de executare repetată se termină atunci când valoarea lui **a** este 0. În al doilea exemplu, se compară valoarea contorului **i** cu numărul total de executări repetate **n** (*i<=n*) și se continuă ciclarea atât timp cât operatorul relațional furnizează valoarea „adevărat". Procesul de executare repetată se termină atunci când valoarea lui **i** este mai mare ca **n**.

Modificarea. Schimbă starea curentă astfel încât să se avanseze către starea finală, care încheie procesul de repetare. Acțiunea de modificare face parte din corpul ciclului. În primul exemplu, modificarea constă în citirea unei noi valori a lui **a** (*citește a*), care poate să fie 0, iar în exemplul al doilea modificarea constă în incrementarea cu 1 a contorului **i** (*i ← i+1*), prin care se evidențiază faptul că s-a mai citit un număr din cele **n** numere și că ne apropiem de citirea ultimului număr.

Inițializarea și modificarea sunt foarte importante, deoarece prin ele trebuie să se ajungă la condiția de terminare. De exemplu, dacă ciclarea se încheie atunci când o variabilă de memorie **i** are valoarea **20**, și dacă inițializăm variabila de memorie **i** cu valoarea **0** și o modificăm prin incrementare cu **3**, ea nu va avea niciodată valoarea **20**, necesară terminării executării repetate a corpului ciclului: 0, 3, 6, 9, 12, 15, 18, 21, 24 etc. Un astfel de ciclu se numește **ciclu infinit** și presupune executarea unui număr infinit de pași, condiție care contrazice definiția unui algoritm.

Observaţi că cele două exemple diferă prin **informaţia pe care o avem referitor la numărul de executări repetate ale ciclului**. În primul exemplu, nu se cunoaşte de câte ori se execută structura repetitivă (condiţia de terminare este valoarea lui *a* şi nu se ştie câte valori vor fi introduse pentru *a* până când *a* va avea valoarea 0). În al doilea exemplu, se cunoaşte de câte ori se execută structura repetitivă (condiţia de terminare este ca valoarea lui *i*, care porneşte iniţial de la valoarea 1, să ajungă printr-o incrementare cu 1 la o valoare *n*, deci să se execute de *n* ori corpul ciclului). În funcţie de acest criteriu, există:

- **structuri repetitive cu număr necunoscut de paşi**,
- **structuri repetitive cu număr cunoscut de paşi**.

Structura repetitivă cu număr cunoscut de paşi

În cazul **structurilor repetitive cu număr cunoscut** de paşi sunt necesare două variabile de memorie: una numită **contor** (în exemplu, **i**), care se foloseşte pentru a număra de câte ori s-a executat repetat corpul ciclului, şi una numită **număr total de repetări** (în exemplu, **n**), care determină de câte ori trebuie să se execute repetat corpul ciclului. În acest caz, condiţia de executare repetată a corpului ciclului este ca valoarea contorului să fie mai mică, sau cel mult egală cu valoarea numărului total de repetări. Iniţializarea ciclului trebuie să conţină obligatoriu şi pasul de iniţializare a contorului, iar modificarea trebuie să conţină obligatoriu operaţia de incrementare a contorului. Observaţi că acest tip de structură poate fi descris prin raţionamentul **pentru... de la... până la... crescând cu... execută... sfârşit (for... to... by... do... endfor)** pe care îl putem folosi în pseudocod, pentru descrierea structurii.

```
pentru contor← vi, vf [pas v] execută
    acţiune;
sfârşit_pentru;
```

→

```
for contor ← vi, vf [by v] do
    acţiune;
endfor;
```

Cuvântul **pentru (for)** marchează începutul structurii, iar cuvântul **sfârşit_pentru (endfor)**, sfârşitul structurii. Virgula dintre *contor* ← *vi* şi *vf* delimitează operaţia de atribuire prin care se face iniţializarea contorului (*vi*), de valoarea finală la care trebuie să ajungă contorul (*vf*), cuvântul **pas (by)** precede valoarea cu care se incrementează contorul (*v*), iar cuvântul **execută (do)** marchează începutul secvenţei de paşi care se va executa repetat. Dacă incrementarea contorului se face cu valoarea 1, expresia **pas** *v* (**by** *v*) este implicită şi nu mai trebuie menţionată.

```
întreg i,n;
real   s,a;
început
 citeşte n;
 s ← 0;
 pentru i←1,n execută
    citeşte a;
    s ← s+a;
 sfârşit_pentru;
 scrie s;
sfârşit.
```

Exemplul 2 a fost transpus în pseudocod.

Probleme care se pot rezolva cu ajutorul structurii repetitive cu număr cunoscut de paşi:

1. Se citesc două numere naturale **m** şi **n**. Calculaţi $\mathbf{n^m}$.

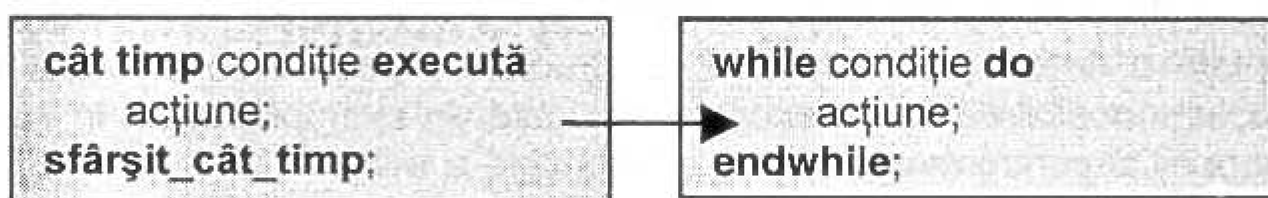
2. Se citesc de la tastatură **n** numere întregi. Să se afle câte numere sunt negative.
3. Se citesc de la tastatură **n** numere întregi. Să se calculeze media aritmetică a numerelor pare.
4. Să se afişeze primele **n** numere naturale divizibile cu 5.
5. S-a depus într-un cont la o bancă o sumă de **s** unităţi monetare. Banca practică următoarele dobânzi: **d1** – lunar, **d2** – trimestrial, **d3** – semestrial şi **d4** – anual. Calculaţi ce sumă va fi în cont după **n** ani, dacă se capitalizează dobânda, în toate cele patru cazuri. Ce dobândă este mai avantajoasă?
6. Se citesc două numere întregi **a** şi **b**. Să se calculeze produsul **a** × **b** fără a folosi operatorul pentru înmulţire (**Indicaţie**. Rezultatul se va obţine prin adunarea repetată a lui |**a**| de |**b**| ori şi se va face discuţie după semnul operanzilor).
7. Se citesc două numere întregi **a** şi **b**. Să se calculeze câtul şi restul împărţirii lui **a** la **b**, fără a folosi operatorii **mod** şi **div** (**Indicaţie**. Rezultatul se va obţine prin scăderea repetată a lui |**b**| din |**a**| şi se va face discuţie după semnul operanzilor).
8. Se citesc de la tastatură **n** numere întregi nenule. Să se calculeze suma celor de rang par şi produsul celor de rang impar.
9. Se citeşte un număr natural **n**. Să se determine toate perechile de numere naturale **a** şi **b** care verifică relaţia $a^2 + b^2 = n$.
10. Calculaţi suma:
    S = 1x3 + 2x5 + 3x7 + ... + nx(2n+1)
    unde n∈N, valoarea lui introducându-se de la tastatură.
11. Calculaţi suma:
    S = 1 + 1x2 + 1x2x3 + ... + 1x2x3x...xn
    unde n∈N, valoarea lui introducându-se de la tastatură.
12. Calculaţi suma:
    $S = 1^2 - 2^2 + 3^2 - 4^2 + \ldots\ldots + (-1)^{n+1} \times n^2$
    unde n∈N, valoarea lui introducându-se de la tastatură.
13. Calculaţi suma:
    $S = 1 - a + a^2 - a^3 + a^4 + \ldots\ldots + (-1)^n \times a^n$
    unde a∈R şi n∈N, valorile lor introducându-se de la tastatură.

Structura repetitivă cu număr necunoscut de paşi

În funcţie de momentul în care se face testarea, există două tipuri de **structuri repetitive cu număr necunoscut de paşi**:

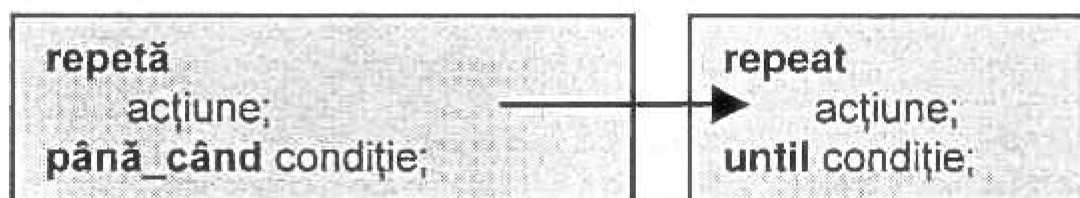
- **structuri repetitive condiţionate anterior,**
- **structuri repetitive condiţionate posterior.**

Structurile repetitive condiţionate anterior testează condiţia de terminare a ciclului înainte de executarea corpului ciclului. Acest tip de structură poate fi descris prin raţionamentul **cât timp... execută... sfârşit (while... do... endwhile)** pe care îl putem folosi în pseudocod pentru descrierea structurii.



Cuvântul **cât timp** (**while**) marchează începutul structurii, iar cuvântul **sfârşit_cât_timp** (**endwhile**), sfârşitul structurii. Cuvintele **cât timp** (**while**) şi **execută** (**do**) delimitează expresia logică ce se evaluează, pentru a testa condiția de terminare a ciclului (corpul ciclului se execută dacă expresia logică are valoarea „adevărat") iar cuvântul **execută** (**do**) marchează începutul secvenței de paşi care se va executa repetat.

Structurile repetitive condiționate posterior testează condiția de terminare a ciclului, după executarea corpului ciclului. Acest tip de structură poate fi descris prin raționamentul **repetă... până când...** (**repeat... until...**) pe care îl putem folosi în pseudocod, pentru descrierea structurii.



Cuvântul **repetă** (**repeat**) marchează începutul structurii, iar sfârşitul expresiei logice marchează sfârşitul structurii. Cuvintele **repetă** (**repeat**) şi **până_când** (**until**) delimitează secvența de paşi care se va executa repetat, iar cuvântul **până_când** (**until**) precede expresia logică ce se evaluează, pentru a testa condiția de terminare a ciclului (ciclul se termină când condiția logică are valoarea „adevărat"; altfel spus, corpul ciclului se execută cât timp expresia logică are valoarea „fals").

Acelaşi algoritm iterativ poate fi descris prin ambele tipuri de structuri repetitive. Exemplul 1 a fost transpus în pseudocod şi schemă logică prin cele două tipuri de structuri.

Structura repetitivă condiționată anterior

```
real s,a;
început
  s ← 0;
  citeşte a;
  cât timp a<>0 execută
  |  s ← s+a;
  |  citeşte a;
  sfârşit_cât_timp;
  scrie s;
sfârşit.
```

Structura repetitivă condiționată posterior

```
real s,a;
început
  s ← 0;
  citeşte a;
  repetă
  |  s ← s+a;
  |  citeşte a;
  până_când a=0;
  scrie s;
sfârşit.
```

Observații:

1. Pentru acelaşi algoritm, cele două condiții de testare ale terminării executării repetate folosite de cele două tipuri de structuri repetitive sunt complementare. În exemplu, **not** (a<>0) = (a=0). Dacă notăm cu *c1* condiția structurii **cât timp** şi cu *c2* condiția structurii **repetă**, atunci:

$$\textbf{not}\,(c1) = c2$$

2. În cazul structurii condiționate posterior, este obligatoriu să se execute acțiunea cel puțin o dată, spre deosebire de structura repetitivă condiționată anterior, unde este posibil să nu se execute niciodată (în exemplu, cazul în care prima valoare citită pentru *a* este 0).

Concluzii:

Orice structură repetitivă condiționată anterior poate fi transformată într-o structură repetitivă condiționată posterior. Reciproca este şi ea adevărată.

Dacă vom considera că în corpul ciclului există două tipuri de acțiuni: una de modificare, care schimbă starea curentă astfel încât să se avanseze către starea finală (*acțiune_modificare*), şi una prin care se efectuează prelucrările obişnuite (*acțiune_prelucrare*), atunci următoarele două structuri sunt echivalente:

Structura repetitivă condiționată anterior	Structura repetitivă condiționată posterior
acțiune_inițializare; **cât timp** *condiție* **execută** *acțiune_prelucrare;* *acțiune_modificare;* **sfârşit_cât_timp**;	*acțiune_inițializare;* **repetă** *acțiune_prelucrare;* *acțiune_modificare;* **până_când not** *condiție;*

Orice structură repetitivă cu număr cunoscut de paşi poate fi transformată într-o structură repetitivă cu număr necunoscut de paşi. Reciproca nu este întotdeauna adevărată.

Structura repetitivă cu număr cunoscut de paşi	Structura repetitivă cu număr necunoscut de paşi, condiționată anterior
pentru *contor* ← vi,vf **execută** *acțiune_prelucrare;* **sfârşit_pentru**;	*contor* ← vi; **cât timp** *contor*<=vf *acțiune_prelucrare;* *contor* ← *contor+1;* **sfârşit_cât_timp**;

Probleme care se pot rezolva cu ajutorul structurii repetitive cu număr necunoscut de paşi:

1. Se citesc mai multe numere, până când ultimul număr citit este zero. Să se afle câte numere sunt pozitive şi câte numere sunt negative.
2. Se citesc mai multe numere întregi, până când ultimul număr citit este zero. Să se calculeze media aritmetică a numerelor impare.
3. Se citeşte un număr natural **n**. Să se afişeze toate numerele naturale mai mici decât *n* care sunt divizibile cu 3.
4. Se citesc mai multe numere întregi, până când ultimul număr citit este zero. Să se calculeze suma celor de rang par şi produsul celor de rang impar.

3.3. Algoritmi elementari

Ați văzut că cea mai importantă etapă în rezolvarea unei probleme cu ajutorul calculatorului este cea de stabilire a algoritmului, deci de găsire a metodei de rezolvare a problemei a cărei soluție urmează să fie calculată. Pentru a vă ușura munca de descoperire a algoritmului, puteți să vă folosiți de **algoritmi elementari**. Acești algoritmi oferă metode de rezolvare pentru probleme clasice:

- ✓ **algoritmi pentru interschimbarea valorilor a două numere;**
- ✓ **algoritmi pentru determinarea valorii minime (maxime);**
- ✓ **algoritmi pentru prelucrarea cifrelor unui număr;**
- ✓ **algoritmi pentru determinarea c.m.m.d.c. dintre două numere;**
- ✓ **algoritmi pentru testarea numerelor prime;**
- ✓ **algoritmi pentru prelucrarea divizorilor unui număr;**
- ✓ **algoritmi pentru conversii între sisteme de numerație;**
- ✓ **algoritmi pentru generarea șirurilor recurente.**

Cu ajutorul acestor algoritmi, puteți prelucra un număr sau o mulțime de valori numerice, pentru a obține informații.

3.3.1. Algoritmi pentru interschimbare

Interschimbarea valorilor a două variabile de memorie *x* și *y* nu se poate face prin simpla atribuire a noii valori, deoarece secvența de atribuiri x←y și y←x duce la pierderea valorii lui *x*, iar secvența de atribuiri y←x și x←y duce la pierderea valorii lui *y*. Pentru a realiza interschimbarea puteți folosi una dintre următoarele variante de algoritmi:

Varianta 1. Interschimbarea valorilor a două variabile (*a* și *b*) **prin folosirea unei variabile intermediare** (*x*). Variabila intermediară se folosește pentru salvarea valorii care se distruge prin prima operație de atribuire. Pașii algoritmului sunt:

Pasul 1. *Se salvează valoarea primei variabile (a) în variabila x, prin x ← a.*

Pasul 2. *Se atribuie primei variabile, a cărei valoare a fost salvată (a), valoarea celei de a doua variabile (b), prin a ← b.*

Pasul 3. *Se atribuie celei de a doua variabile (b) valoarea primei variabile, care a fost salvată (x) prin b ← x.*

Varianta 2. Interschimbarea valorilor a două variabile (*a* și *b*) **fără folosirea unei variabile intermediare**. Se folosesc identitățile matematice **a=(a-b)+b** și **b=((a-b)+b)-(a-b)**. Pentru interschimbarea valorilor se folosește valoarea *a-b*, care va fi atribuită inițial variabilei *a*.

Varianta 1

```
real a,b,x;
început
 citeşte a,b;
 x ← a;  a ← b;  b ← x;
 scrie a,b;
sfârşit.
```

Varianta 2

```
real a,b;
început
 citeşte a,b;
 a ← a-b; b ← a+b; a ← b-a;
 scrie a,b;
sfârşit.
```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul de interschimbare în rezolvarea unei probleme.

Enunțul problemei: *Se citește un număr natural format din 3 cifre. Să se afișeze numărul minim care se poate forma din cifrele sale.* De exemplu, dacă numărul este 312, numărul minim care se poate forma cu cifrele sale este 123.

Algoritmul constă în executarea următorilor pași:

Pasul 1. *Se extrag cifrele numărului.*
Pasul 2. *Se ordonează crescător cifrele numărului.*
Pasul 3. *Se formează un nou număr din cifrele ordonate crescător.*

Pentru ordonarea crescătoare a cifrelor, se va rafina **Pasul 2**. **Procesul de rafinare** constă în descompunerea unei probleme în subprobleme. Considerând cifrele numărului *a* (cifra sutelor), *b* (cifra zecilor) și *c* (cifra unităților), pentru a le ordona le vom compara două câte două și, dacă nu respectă relația de ordine precizată, le interschimbăm:

Pasul 2.1. *Se compară* ***a*** *cu* ***b****. Dacă* ***a*** *este mai mare decât* ***b****, se interschimbă* ***a*** *cu* ***b****.*
Pasul 2.2. *Se compară* ***b*** *cu* ***c****. Dacă* ***b*** *este mai mare decât* ***c****, se interschimbă* ***b*** *cu* ***c****.*
Pasul 2.3. *Se compară* ***a*** *cu* ***b****. Dacă* ***a*** *este mai mare decât* ***b****, se interschimbă* ***a*** *cu* ***b****.*

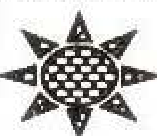
Se vor folosi următoarele variabile de memorie:

- ✓ **Data de intrare** *n* pentru număr.
- ✓ **Date intermediare** *a*, *b* și *c* pentru cifrele numărului, și *x* pentru operația de interschimbare.
- ✓ **Data de ieșire** *n* pentru noul număr (putem folosi aceeași variabilă de memorie deoarece nu mai avem nevoie de vechea valoare a numărului).

Algoritmul este:

```
întreg n,a,b,c,x;
început
 citeşte n;
 a ← n div 100; b ← (n div 10) mod 10; c ← n mod 10;
 dacă a>b
    atunci x ← a; a ← b;  b ← x;
 sfârşit_dacă;
 dacă b>c
    atunci x ← b; b ← c;  c ← x;
 sfârşit_dacă;
 dacă a>b
    atunci x ← a; a ← b;  b ← x;
 sfârşit_dacă;
 n ← a*100+b*10+c;
 scrie n;
sfârşit.
```

Pentru testarea algoritmului, folosiți următoarea mulțime de valori pentru data de intrare **n**: {123, 132, 213, 231, 312, 321}. Testați algoritmul. Refaceți algoritmul folosind metoda de interschimbare care nu utilizează o variabilă intermediară. Testați algoritmul.



Probleme care se pot rezolva cu ajutorul algoritmului de interschimbare:

1. Se citeşte un număr natural format din 3 cifre. Să se afişeze numărul maxim care se poate forma din cifrele sale.
2. Se citeşte un număr natural format din 4 cifre. Să se afişeze numărul maxim, respectiv numărul minim care se poate forma din cifrele sale.

3.3.2. Algoritmi pentru determinarea maximului (minimului)

Algoritmul determină valoarea maximă (minimă) dintr-un şir de numere introduse de la tastatură. Algoritmul constă în atribuirea valorii primului element maximului (minimului) şi compararea acestei valori cu elementele din şir. Paşii care se execută sunt:

Pasul 1. *Se citeşte primul număr **a**.*
Pasul 2. *Se atribuie maximului valoarea lui **a**, prin max ←a.*
Pasul 3. *Se citeşte următorul număr **a**.*
Pasul 4. *Dacă a>max, atunci se atribuie maximului valoarea lui **a**, prin max←a.*
Pasul 5. *Dacă mai sunt date de citit se revine la **Pasul 3**.*

Se vor folosi următoarele variabile de memorie:
✓ **Data de intrare** *a* pentru citirea unui număr.
✓ **Data de ieşire** *max* pentru valoarea maximă.

Varianta 1. Se introduce un şir de *n* numere de la tastatură. Să se afişeze maximul dintre aceste numere.

Varianta 2. Se introduce un şir de numere de la tastatură, până la întâlnirea valorii *0*. Să se afişeze maximul dintre aceste numere.

Varianta 1
```
întreg a,max,n,i;
început
  citeşte n,a; max ← a;
  pentru i ← 2,n execută
     citeşte a;
     dacă a>max
       atunci max ← a;
     sfârşit_dacă;
  sfârşit_pentru;
  scrie max;
sfârşit.
```

Varianta 2
```
întreg a,max;
început
  citeşte a; max ← a;
  cât timp a<>0 execută
     dacă a>max
       atunci max ← a;
     sfârşit_dacă;
     citeşte a;
  sfârşit_cât_timp;
  scrie max;
sfârşit.
```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru determinarea valorii maxime (minime).

Enunţul problemei: *Se introduc de la tastatură **n** numere. Să se afişeze valoarea maximă şi de câte ori apare aceasta în şir.*

Pentru numărarea apariţiilor maximului, se foloseşte variabila de memorie **k**.

Pasul 1. *Se citeşte primul număr **a**.*
Pasul 2. *Se atribuie maximului valoarea lui **a**, prin max←a, şi se iniţializează contorul **k**, prin k←1.*
Pasul 3. *Se citeşte următorul număr **a**.*
Pasul 4. *Dacă a=max, atunci se incrementează contorul **k**, prin k←k+1.*
Pasul 5. *Dacă a>max, atunci se atribuie maximului valoarea lui **a**, prin max←a, şi se iniţializează contorul **k**, prin k←1.*
Pasul 6. *Dacă mai sunt date de citit se revine la **Pasul 3**.*

```
întreg a,max,n,i,k;
început
 citeşte n,a;
 max ← a; k ← 1;
 pentru i ← 2,n execută
   citeşte a;
   dacă a=max
     atunci k ← k+1;
     altfel dacă a>max
               atunci max ← a; k ← 1;
            sfârşit_dacă;
   sfârşit_dacă;
 sfârşit_pentru;
 scrie max,k;
sfârşit.
```



Probleme care se pot rezolva cu ajutorul algoritmilor de determinare a minimului (maximului)

1. Se introduc de la tastatură *n* numere. Să se afişeze valoarea minimă şi valoarea maximă.
2. Se introduce un şir de numere de la tastatură, până la întâlnirea valorii *0*. Să se afişeze maximul şi minimul dintre aceste numere.
3. La un concurs, comisia de notare este formată din *n* membri. Să se scrie algoritmul de calcul al mediei, ştiind că nota cea mai mică şi nota cea mai mare nu sunt luate în considerare la calcularea mediei.
4. Se introduce un şir de numere de la tastatură, până la întâlnirea valorii *0*. Să se afişeze valoarea maximă şi de câte ori apare în şir.
5. Se introduce un şir de numere întregi de la tastatură, până la întâlnirea valorii *0*. Să se afişeze:
   a) maximul dintre numerele negative;
   b) minimul dintre numerele negative;
   c) maximul dintre numerele pozitive;
   d) minimul dintre numerele pozitive.
6. Se introduce un şir de *n* numere întregi, de la tastatură. Să se afişeze:
   a) maximul dintre numerele negative;
   b) minimul dintre numerele negative;
   c) maximul dintre numerele pozitive;
   d) minimul dintre numerele pozitive.
7. Se introduc de la tastatură *n* numere întregi. Să se afişeze primele două valori maxime (sau, varianta, minime) şi de câte ori apar în şirul de numere.

3.3.3. Algoritmi pentru prelucrarea cifrelor unui număr

Pentru prelucrarea cifrelor unui număr puteți folosi următorii algoritmi;

- ✓ **algoritmul pentru extragerea cifrelor unui număr;**
- ✓ **algoritmul pentru compunerea unui număr din cifrele sale;**
- ✓ **algoritmul pentru determinarea inversului unui număr (inversarea cifrelor unui număr).**

Algoritmul pentru extragerea cifrelor unui număr

Algoritmul determină cifrele unui număr *n*, prin extragerea pe rând a fiecărei cifre *c* (începând cu cifra unităților), cu operația *n* ***mod*** *10,* și eliminarea din număr a cifrei extrase, cu operația *n* ***div*** *10.* Aceste operații se execută cât timp mai există cifre de extras din *n* (***n<>0***). Pașii care se execută sunt:

Pasul 1. *Se extrage cifra cea mai nesemnificativă, cu operația c ← n* ***mod*** *10.*
Pasul 2. *Se afișează (se prelucrează) cifra.*
Pasul 3. *Se elimină din număr cifra extrasă, cu operația n ← n* ***div*** *10.*
Pasul 4. *Dacă* ***n<>0****, atunci se revine la* ***Pasul 1****.*

Se vor folosi următoarele variabile de memorie:

✓ **Data de intrare** *n* pentru numărul care se citește.
✓ **Data de ieșire** *c* pentru cifra numărului.

```
întreg n,c;
început
 citeşte n;
 cât timp n<>0 execută
    c ← n mod 10; scrie c; n ← n div 10;
 sfârşit_cât_timp;
sfârşit.
```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru extragerea cifrelor unui număr.

Enunțul problemei 1: *Se citește un număr natural* ***n****. Să se afișeze suma și produsul cifrelor sale.*

Se vor folosi următoarele variabile de memorie;

✓ **Data de intrare**: *n* pentru numărul care se citește.
✓ **Date de ieșire**: *s* pentru suma cifrelor și *p* pentru produsul cifrelor numărului.

Enunțul problemei 2: *Se introduce de la tastatură un șir de numere naturale, până la citirea valorii 0. Să se afișeze toate perechile de numere introduse consecutiv care au proprietatea că al doilea număr este egal cu suma cifrelor primului număr.*

Se vor folosi următoarele variabile de memorie:

- ✓ **Date de intrare**: *a* şi *b* pentru perechile de numere citite consecutiv.
- ✓ **Date intermediare:** *s* pentru calcularea sumei cifrelor şi *x* pentru salvarea valorii lui *a*.
- ✓ **Date de ieşire**: perechile de numere *a* şi *b* care îndeplinesc condiţia.

Algoritmii sunt:

Problema 1

```
întreg n,s,p;
început
 citeşte n;
 s ← 0; p ← 1;
 cât timp n<>0 execută
    s ← s+n mod 10;
    p ← p*(n mod 10);
    n ← n div 10;
 sfârşit_cât_timp;
scrie s,p;
sfârşit.
```

Problema 2

```
întreg a,b,s;
început
 citeşte a,b;
 cât timp b<>0 execută
    s ← 0; x ← a;
    cât timp x<>0 execută
      s ← s+x mod 10;
      x ← x div 10;
    sfârşit_cât_timp;
    dacă s=b
      atunci scrie a,b;
    sfârşit_dacă;
    a ← b;
    citeşte b;
 sfârşit_cât_timp;
sfârşit.
```

Alegeţi pentru testarea algoritmilor câte o mulţime completă de seturi de date de intrare. Testaţi algoritmii.



Algoritmul pentru compunerea unui număr din cifrele sale

Citirea cifrelor numărului se face începând cu cifra cea mai semnificativă. Algoritmul foloseşte reprezentarea numărului în baza 10:

$$a_n a_{n-1} \ldots a_1 a_0 = a_n \times 10^n + a_{n-1} \times 10^{n-1} + \ldots + a_1 \times 10^1 + a_0 \times 10^0$$

Varianta 1. Se introduc pe rând de la tastatură cele *n* cifre ale unui număr începând cu cifra cea mai semnificativă. Să se afişeze numărul *nr* obţinut din aceste cifre. Paşii algoritmului sunt:

Pasul 1. *Se citeşte numărul de cifre* ***n***.

Pasul 2. *Se calculează* ***p*** *ca fiind* 10^{n-1}, *astfel:*

- **Pasul 2.1.** *Se iniţializează* ***p*** *cu valoarea 1, prin operaţia p ← 1.*
- **Pasul 2.2.** *Se iniţializează contorul* ***i***, *cu care se numără puterea lui 10, cu valoarea 1, prin operaţia i ← 1.*
- **Pasul 2.3.** *Se înmulţeşte* ***p*** *cu valoarea 10, prin operaţia p ← p*10.*
- **Pasul 2.4.** *Se incrementează contorul* ***i*** *cu 1, prin operaţia i ← i+1.*
- **Pasul 2.5.** *Dacă i<=n-1 se revine la* ***Pasul 2.3***.

Pasul 3. *Se iniţializează numărul* ***nr*** *cu valoarea 0, prin operaţia nr ← 0.*

Pasul 4. *Se iniţializează contorul* ***i***, *cu care se numără cifrele citite, cu 1, prin i ←1.*

Pasul 5. *Se citeşte cifra* ***c***.

Pasul 6. *Se adună la numărul* ***nr*** *cifra* ***c*** *înmulţită cu puterea lui 10 corespunzătoare poziţiei ei în număr, prin operaţia nr ← nr+c*p.*

Pasul 7. *Se decrementează puterea lui 10, prin operaţia p ← p* ***div*** *10.*

Pasul 8. *Se incrementează contorul **i** cu 1, prin operația de atribuire i ← i+1.*
Pasul 9. *Dacă i<=n, atunci se revine la **Pasul 5**.*
Pasul 10. *Se afişează numărul **nr**.*

Varianta 2. Se introduc pe rând de la tastatură mai multe numere, care reprezintă cifrele unui număr, începând cu cifra cea mai semnificativă, până când se introduce un număr care nu poate fi cifră. Să se afişeze numărul *nr* obținut din aceste cifre. Pentru rezolvarea problemei, algoritmul foloseşte următorul mod de grupare a termenilor reprezentării numărului în baza 10:

$$a_n a_{n-1} \ldots a_1 a_0 = (\ldots((a_n \times 10 + a_{n-1}) \times 10 + a_{n-2}) \times 10 + \ldots + a_1) \times 10 + a_0$$

Paşii care se execută sunt:

Pasul 1. *Se inițializează numărul **nr** cu valoarea 0, prin operația nr ← 0.*
Pasul 2. *Se citeşte cifra **c**.*
Pasul 3. *Se adună la numărul **nr** înmulțit cu 10 cifra citită, prin nr ← nr*10+c.*
Pasul 4. *Se citeşte cifra **c**.*
Pasul 5. *Dacă **c** este o cifră (c>=0 **and** c<=9), atunci se revine la **Pasul 3**.*
Pasul 6. *Se afişează numărul **nr**.*

Varianta 1

```
întreg n,p,nr,i,c;
început
 citeşte n;
 p ← 1;
 pentru i ← 1,n-1 execută
   p ← p*10;
 sfârşit_pentru;
 nr ← 0;
 pentru i ← 1,n execută
  citeşte c;
  nr ← nr + c*p;
  p ← p div 10;
 sfârşit_pentru;
 scrie nr;
sfârşit.
```

Varianta 2

```
întreg c,nr;
început
 nr ← 0;
 citeşte c;
 cât timp c>=0 and c<=9 execută
   nr ← nr*10 + c;
   citeşte c;
 sfârşit_cât_timp;
 scrie nr;
sfârşit.
```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru compunerea unui număr din cifrele sale.

Enunțul problemei: *Se citesc mai multe numere, care reprezintă cifrele unui număr binar, până când numărul citit nu mai este cifră binară. Să se afişeze numărul binar.* De exemplu, dacă se introduc numerele 1, 0, 1, 1, 5, numărul binar afişat va fi **1011**.

Pentru a scrie pe ecran numărul obținut, se va folosi scrierea lui ca un număr în baza 10 (cu cifrele corespunzătoare reprezentării în baza *2*).

```
întreg c,nr;
început
 nr ← 0;
 citeşte c;
 cât timp c=0 or c=1 execută
   nr ← nr*10 + c; citeşte c;
 sfârşit_cât_timp
 scrie nr;
sfârşit.
```



Algoritmul pentru determinarea inversului unui număr

Algoritmul determină *inv* inversul numărului *n* prin extragerea pe rând a fiecărei cifre (începând cu cifra unităţilor) din numărul *n* şi compunerea unui nou număr cu aceste cifre. De exemplu, dacă numărul este 123, inversul va fi 321. Paşii algoritmului sunt:

Pasul 1. *Se citeşte numărul **n**.*
Pasul 2. *Se iniţializează numărul invers **inv** cu valoarea 0, prin operaţia inv ← 0.*
Pasul 3. *Se extrage cifra cea mai nesemnificativă din numărul **n** şi se adună cifra la numărul **inv** înmulţit cu 10, prin operaţia inv ← inv*10 + n **mod** 10.*
Pasul 4. *Se elimină din numărul n cifra extrasă, cu operaţia n ← n **div** 10.*
Pasul 5. *Dacă **n<>0**, atunci se revine la **Pasul 3**.*

```
întreg n,inv;
început
 citeşte n;
 inv ← 0;
 cât timp n<>0 execută
   inv ← inv*10 + n mod 10;  n ← n div 10;
 sfârşit_cât_timp;
 scrie inv;
sfârşit.
```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru determinarea inversului unui număr.

Enunţul problemei: *Se citeşte un număr natural n. Să se verifice dacă este **palindrom** (un număr este palindrom dacă, citit de la stânga la dreapta şi de la dreapta la stânga, are aceeaşi valoare; de exemplu, 12321).*

Deoarece, prin calcularea inversului numărului *n*, valoarea acestuia se pierde prin eliminarea cifrelor, iar după calcularea inversului mai avem nevoie de valoarea lui *n* pentru a o compara cu valoarea inversului, se va folosi o variabilă de memorie *nr*, pentru a salva valoarea lui *n*.

```
întreg n,nr,inv;
început
 citeşte n;
 nr ← n; inv ← 0;
```

```
   cât timp n<>0 execută
      inv ← inv*10 + n mod 10; n ← n div 10;
   sfârşit_cât_timp;
   dacă nr=inv
      atunci scrie "Numarul este palindrom";
      altfel scrie "Numarul nu este palindrom";
   sfârşit_dacă;
sfârşit.
```



Probleme care se pot rezolva cu ajutorul algoritmilor pentru prelucrarea cifrelor unui număr:

1. Se citeşte un număr natural *n*. Să se afişeze suma şi produsul cifrelor pare (sau impare).
2. Se citeşte un număr natural *n*. Să se afişeze suma şi produsul cifrelor din poziţiile pare (sau, variantă, impare). Numărarea poziţiilor se face începând cu cifra cea mai semnificativă.
3. Se introduc de la tastatură *n* numere. Să se afişeze cea mai mare cifră a fiecărui număr.
4. Să se afişeze toate numerele naturale care au proprietatea că sunt egale cu pătratul sumei cifrelor lor (**Indicaţie**. Se demonstrează, matematic, că un astfel de număr nu poate avea decât maxim 4 cifre. Exemplu: 81⇒ 8+1=9; 81= 9×9).
5. Se citesc *n* numere naturale. Să se afişeze, pentru fiecare număr din şir, numărul obţinut prin eliminarea tuturor cifrelor 0.
6. Să se afişeze toate numerele care sunt palindrom şi care aparţin intervalului [*a*,*b*]. Valorile pentru *a* şi *b* se citesc de la tastatură.
7. Se citeşte un şir de *n* numere naturale. Să se afişeze cele care sunt palindroame.
8. Să se afişeze toate numerele din intervalul [*a*,*b*] care au suma cifrelor un număr par. Valorile pentru *a* şi *b* se citesc de la tastatură.
9. Se citesc de la tastatură un număr $k \neq 0$ şi un şir de numere întregi, până la întâlnirea numărului 0. Să se afişeze câte numere din şir au suma cifrelor *k*.
10. Se introduce de la tastatură un şir de numere naturale, până la citirea numărului 0. Să se afişeze toate tripletele de numere introduse consecutiv care au proprietatea că al doilea şi al treilea număr sunt egale cu câtul, respectiv cu restul dintre împărţirea primului număr la suma cifrelor sale.
11. Se citesc de la tastatură un număr $k \in [0,9]$ şi un şir de numere naturale, până la citirea numărului 0. Să se afişeze toate perechile de numere introduse consecutiv care au proprietatea că au acelaşi număr de apariţii ale cifrei *k* în pătratul lor.
12. Se introduce de la tastatură un şir de numere naturale, până la citirea numărului 0. Să se afişeze toate perechile de numere introduse consecutiv care au proprietatea că suma cifrelor primului număr este pară, iar suma cifrelor celui de al doilea număr este impară.

3.3.4. Algoritmi pentru calcularea c.m.m.d.c.

Pentru calcularea c.m.m.d.c dintre două numere naturale nenule, se folosesc următorii algoritmi:

Varianta 1. Foloseşte **algoritmul lui Euclid** care atribuie lui *b* **restul împărţirii** lui *a* la *b*, iar lui *a* îi atribuie vechea valoare a lui *b*. Rezolvarea problemei se bazează pe condiţia b≠0. Paşii algoritmului sunt:

Pasul 1. *Se împarte **a** la **b** şi se obţine restul **r** (r← a mod b).*
Pasul 2. *Se execută operaţiile de atribuire a←b; b←r.*
Pasul 3. *Dacă b<>0, atunci se revine la **Pasul 1**, altfel **cmmdc**←a.*

De exemplu, dacă a=18 şi b=12, calculul se desfăşoară astfel:

1. Se calculează restul: r = 18 mod 12 = 6.
2. Se fac atribuirile: a=12, b=6.
3. b≠0 (6≠0) ⇒ Se calculează restul: r = 12 mod 6 = 0.
4. Se fac atribuirile: a=6, b=0.
5. b=0 (0=0) ⇒ cmmdc = a = 6.

Verificaţi algoritmul şi pentru a=12 şi b=18.

Varianta 2. Foloseşte **algoritmul prin scădere repetată** a valorii celei mai mici din valoarea cea mai mare. Rezolvarea problemei se bazează pe condiţia a≠b. Paşii care se execută sunt:

Pasul 1. *Se scade din numărul mai mare celălalt număr: dacă a>b, se execută operaţia a←a-b, altfel se execută operaţia b←b-a.*
Pasul 2. *Dacă a<>b, atunci se revine la **Pasul 1**; altfel, **cmmdc**←a.*

De exemplu, dacă a=18 şi b=12, calculul se desfăşoară astfel:

1. a>b ⇒ Se face atribuirea: a=18-12=6.
2. b>a ⇒ Se face atribuirea: b=12-6=6.
3. a=b (6=6) ⇒ cmmdc = a = 6.

Varianta 1

```
întreg a,b,r;
început
 citeşte a,b;
 cât timp b<>0 execută
    r ← a mod b;
    a ← b;
    b ← r;
 sfârşit_cât_timp;
 scrie "cmmdc=",a;
sfârşit.
```

Varianta 2

```
întreg a,b;
început
 citeşte a,b;
 cât timp a<>b execută
     dacă a>b
        atunci a ← a-b;
        altfel b ← b-a;
     sfârşit_dacă
 sfârşit_cât_timp;
 scrie "cmmdc=",a;
sfârşit.
```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru calcularea c.m.m.d.c. a două numere.

Enunțul problemei: *Să se calculeze cel mai mic multiplu comun și cel mai mare divizor comun a două numere naturale **a** și **b** care se introduc de la tastatură.*

Calcularea c.m.m.m.c. a două numere se bazează pe calculul c.m.m.d.c. Dacă notăm cmmdc(a,b) cu **c** și cmmmc(a,b) cu **m**, atunci $a = x \times c$, iar $b = y \times c$, unde x și y sunt numere prime între ele. Rezultă că $m = x \times y \times c = (a/c) \times (b/c) \times c = a \times b/c$.

Deoarece prin algoritmul de calcul al c.m.m.d.c. se pierd valorile inițiale ale lui *a* și *b*, ele se vor salva în două variabile de memorie: *x* și *y*.

În cazul în care *a* și *b* au ambele valoarea 0, c.m.m.d.c. nu se poate calcula.

Algoritmul este:

```
întreg a,b,c,r,x,y,m;
început
 citeşte a,b;
 x ← a; y ← b;
 dacă b=0
   atunci c ← a;
   altfel
       cât timp b<>0 execută
            r ← a mod b; a ← b; b ← r;
       sfârşit_cât_timp;
       c ← a;
 sfârşit_dacă;
 dacă c=0
  atunci scrie "Nu se pot calcula; ambele numere sunt 0";
  altfel dacă x=0 or y=0
          atunci scrie "Nu se poate calcula c.m.m.m.c";
                 scrie "c.m.m.d.c.= ",c;
          altfel m ← x*y/c;
                 scrie c,m;
        sfârşit_dacă;
 sfârşit_dacă;
sfârşit.
```



Alegeți pentru testarea algoritmului o mulțime completă de seturi de date de intrare. Testați algoritmul.

Probleme care se pot rezolva cu ajutorul algoritmilor pentru calculul c.m.m.d.c.:

1. Se citesc de la tastatură două numere naturale *n* și *k* (2≤k≤n). Să se afișeze toate perechile de numere naturale mai mici decât *n* al căror c.m.m.d.c. este *k*.
2. Să se scrie algoritmul prin care se calculează c.m.m.d.c. și c.m.m.m.c. a 3 numere introduse de la tastatură. Să se generalizeze problema pentru *n* numere introduse de la tastatură.

3. Să se afişeze toate numerele naturale mai mici decât un număr natural *n* care sunt prime cu *n*, *n* introducându-se de la tastatură (două numere naturale se numesc **prime între ele** dacă cel mai mare divizor comun al lor este 1).

3.3.5. Algoritmi pentru testarea unui număr prim

Algoritmul de verificare a unui număr natural *n*, dacă este prim, constă în generarea tuturor numerelor naturale mai mari sau egale cu 2 şi mai mici sau egale cu **sqrt**(*n*) şi verificarea, pentru fiecare număr generat, dacă îl divide pe *n*. Dacă există cel puţin un astfel de număr, numărul *n* nu este prim. Pentru a şti dacă există cel puţin un număr care îl divide pe *n* se va folosi o variabilă logică *x*, care va avea valoarea *True* dacă numărul este prim şi *False* dacă numărul nu este prim. Se presupune că numărul este prim (variabila *x* se iniţializează cu valoarea *True*) şi, pentru primul număr, găsit în şirul de numere generate, care îl divide pe *n*, se va schimba valoarea variabilei *x* în *False* (numărul nu mai este considerat prim). Pentru generarea şirului de numere se foloseşte o variabilă contor *i*, care va fi iniţializată cu valoarea 2 şi care se va incrementa cu *1* până va avea valoarea [n/2]. Paşii care se execută sunt:

Pasul 1. *Se iniţializează variabila **x**, prin operaţia x ← **T**.*
Pasul 2. *Se generează prima cifră din şirul de numere, prin operaţia i ← 2.*
Pasul 3. *Dacă **n** se divide cu **i**, atunci se schimbă valoarea variabilei x, prin operaţia x ← **F**; altfel, se generează următoarea cifră din şirul de numere, prin incrementarea contorului i ← i+1.*
Pasul 4. *Dacă i<=sqrt(n) (nu s-au generat toate numerele care ar putea fi divizori) şi dacă x=**T** (numărul este considerat în continuare prim), atunci se revine la **Pasul 3.***
Pasul 5. *Dacă x=**T** se afişează mesajul "Numarul este prim"; altfel, se afişează mesajul "Numarul nu este prim".*

```
întreg n,i;
logic x;
început
 citeşte n;
 x ← T; i ← 2;
 cât timp i<=sqrt(n) and x execută
    dacă n mod i = 0
     atunci x ← F;
     altfel i ← i+1;
    sfârşit_dacă;
 sfârşit_cât_timp;
 dacă x
     atunci scrie "Numarul este prim";
     altfel scrie "Numarul nu este prim";
 sfârşit_dacă;
sfârşit.
```

Se observă că algoritmul se poate optimiza prin eliminarea, din şirul generat, al numerelor pare, deoarece, dacă numărul *n* nu se divide prin 2, nu se va divide prin nici un număr par. Algoritmul va fi:

```
întreg n,i;
logic x;
început
 citeşte n; x ← T;
 dacă n mod 2 = 0
   atunci x ← F;
   altfel i ← 3;
          cât timp i<=sqrt(n) and x execută
            dacă n mod i = 0
              atunci x ← F;
              altfel i ← i+2;
            sfârşit_dacă;
          sfârşit_cât_timp;
 sfârşit_dacă;
 dacă x
   atunci scrie "Numarul este prim";
   altfel scrie "Numarul nu este prim";
 sfârşit_dacă;
sfârşit.
```

Problema se poate rezolva şi fără să se folosească variabila *x*, doar prin testarea valorii lui *i*. În cazul în care numărul nu mai este considerat prim, *i* va lua o valoare în afara şirului de valori generate. Algoritmul va fi:

Pasul 1. *Se generează prima cifră din şirul de numere, prin operaţia i ← 2.*

Pasul 2. *Dacă **n** se divide cu **i**, atunci se atribuie lui **i** o valoare în afara şirului de valori generate i ← n; altfel, se generează următoarea cifră din şirul de numere, prin incrementarea contorului i ← i+1.*

Pasul 3. *Dacă i<=sqrt(n), atunci se revine la **Pasul 2**.*

Pasul 4. *Dacă i<>n se afişează mesajul "Numarul este prim"; altfel, se afişează mesajul "Numarul nu este prim".*

```
întreg n,i;
început
 citeşte n;
 dacă n mod 2 = 0
   atunci i ← n;
   altfel i ← 3;
          cât timp i<=sqrt(n) execută
            dacă n mod i = 0
              atunci i ← n;
              altfel i ← i+2;
            sfârşit_dacă;
          sfârşit_cât_timp;
 sfârşit_dacă;
 dacă i<>n
     atunci scrie "Numarul este prim";
     altfel scrie "Numarul nu este prim";
 sfârşit_dacă;
sfârşit.
```

Alegeţi pentru testarea algoritmilor o mulţime completă de seturi de date de intrare. Testaţi algoritmii.

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul prin care se testează un număr dacă este prim.

Enunţul problemei: *Să se afişeze toate numerele prime din intervalul [a,b],* ***a*** *şi* ***b*** *introducându-se de la tastatură.*

Numerele din intervalul [a,b] se vor genera în variabila de memorie *n*. Algoritmul va fi:

Pasul 1. *Se va verifica mai întâi dacă valorile lui a şi b sunt în relaţia de ordine a<b, pentru a reprezenta un interval. Dacă a>b se trece la* ***Pasul 6****.*
Pasul 2. *Se generează primul număr din interval, cu operaţia n ← a.*
Pasul 3. *Se verifică dacă numărul* ***n*** *este număr prim. Dacă este număr prim se afişează.*
Pasul 4. *Se generează următorul număr din interval, cu operaţia n ← n+1.*
Pasul 5. *Se verifică dacă nu s-a ajuns la capătul intervalului: dacă n<b, se revine la* ***Pasul 3****; altfel, se termină algoritmul.*
Pasul 6. *Se afişează mesajul: a, " si ", b," nu formează un interval".*

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare**: *a* şi *b* pentru limitele intervalului.
- ✓ **Date intermediare**: *n*, pentru numărul care se generează, şi *i*, pentru generarea şirului de numere cu care se împarte *n*.
- ✓ **Data de ieşire**: *n*, în cazul în care valoarea sa este un număr prim.

```
întreg a,b,n,i;
început
 citeşte a,b;
 dacă a<b
   atunci pentru n ← a,b execută
            dacă n mod 2 = 0
              atunci i ← n;
              altfel i ← 3;
                     cât timp i<=sqrt(n) execută
                       dacă n mod i = 0
                         atunci i ← n;
                         altfel i ← i+2;
                       sfârşit_dacă;
                     sfârşit_cât_timp;
            sfârşit_dacă;
            dacă i<>n
              atunci scrie n;
            sfârşit_dacă;
          sfârşit_pentru;
   altfel scrie a, " si ", b," nu formează un interval"
 sfârşit_dacă;
sfârşit.
```



Probleme care se pot rezolva cu ajutorul algoritmului de testare a unui număr dacă este prim:

1. Se introduc *n* numere de la tastatură. Să se afişeze numerele prime.
2. Să se afişeze primele *n* numere prime, *n* introducându-se de la tastatură.
3. Să se afişeze primele *n* numere prime care au suma cifrelor mai mică decât un număr *m*, *n* şi *m* introducându-se de la tastatură.
4. Să se afişeze toate numerele prime de patru cifre care au inversul tot număr prim.
5. Să se afişeze descompunerea unui număr natural par, strict mai mare decât 2, într-o sumă de două numere prime (verificarea ipotezei lui Goldbach).
6. Să se afişeze primele *n* perechi de **numere prime gemene**, unde *n* este un număr natural introdus de la tastatură. (Două numere prime *a* şi *b* sunt gemene dacă b-a=2. Exemple: 3 şi 5, 5 şi 7, 11 şi 13, 17 şi 19, 29 şi 31.)
7. Să se afişeze cel mai mare număr prim, mai mic decât un număr dat *n*. (Exemplu: dacă n=10, numărul va fi 7.)
8. Să se afişeze cel mai mic număr prim, mai mare decât un număr dat *n*. (Exemplu: dacă n=10, numărul va fi 11.)

3.3.6. Algoritmi pentru prelucrarea divizorilor unui număr

Algoritmul pentru generarea divizorilor proprii ai unui număr

Algoritmul de generare a **divizorilor proprii** ai unui număr *n* constă în împărţirea numărului la un şir de numere i, i∈[2,[n/2]]. Dacă numărul *n* se împarte la numărul generat, atunci *i* este divizor al lui *n*. Paşii algoritmului sunt:

Pasul 1. *Afişează divizorii 1 şi **n**.*
Pasul 2. *Se iniţializează şirul de numere cu care se va împărţi n cu primul divizor posibil, prin operaţia i ← 2.*
Pasul 3. *Dacă **i** îl divide pe **n**, atunci afişează i.*
Pasul 4. *Se incrementează cu 1 numărul la care se împarte n, prin i ← i+1.*
Pasul 5. *Dacă i<=[n/2], atunci se revine la **Pasul 3**; altfel, se termină algoritmul.*

De exemplu, dacă n=36, afişarea divizorilor proprii se desfăşoară astfel:

1. Afişează **1, 36**.
2. Se iniţializează împărţitorul cu 2, prin operaţia de atribuire i=2.
3. 2 îl divide pe 36 (36 mod 2 =0) ⇒ Afişează **2**.
4. Se incrementează împărţitorul cu 1: i=2+1=3.
5. i<=18 (3<=18) ⇒ 3 îl divide pe 36 (36 mod 3 =0) ⇒ Afişează **3**.
6. Se incrementează împărţitorul cu 1: i=3+1=4.
7. i<=18 (4<=18) ⇒ 4 îl divide pe 36 (36 mod 4 =0) ⇒ Afişează **4**.
8. Se incrementează împărţitorul cu 1: i=4+1=5.
9. i<=18 (5<=18) ⇒ 5 nu îl divide pe 36 (36 mod 5 =1).
10. Se incrementează împărţitorul cu 1: i=5+1=6.
11. i<=18 (6<=18) ⇒ 6 îl divide pe 36 (36 mod 6 =0) ⇒ Afişează **6**.
12. Se incrementează împărţitorul cu 1: i=5+1=7.

....

34. Se incrementează împărţitorul cu 1: i=17+1=18.
35. i<=18 (18<=18) ⇒ 18 îl divide pe 36 (36 mod 18 =0) ⇒ Afişează **18**.
36. Se incrementează împărţitorul cu 1: i=18+1=19.
37. i>18 (19>18) ⇒ Se termină algoritmul.

Se observă că, dacă *i* este un divizor al lui *n*, atunci şi *n/i* este un divizor al lui *n*, şi căutarea divizorilor se poate limita până la radical de ordinul 2 din *n*, obţinându-se varianta a doua a algoritmului, descris prin pseudocod.

Varianta 1

```
întreg n,i;
început
 citeşte n;
 scrie 1, n
 pentru i←2,n mod 2 execută
   dacă n mod i = 0
     atunci scrie i;
   sfârşit dacă;
 sfârşit_pentru;
sfârşit.
```

Varianta 2

```
întreg n,i;
început
 citeşte n;
 scrie 1, n
 pentru i←2,[sqrt(n)] execută
   dacă n mod i = 0
     atunci scrie i, n div i;
   sfârşit dacă;
 sfârşit_pentru;
sfârşit.
```

Algoritmul pentru generarea divizorilor primi ai unui număr

Pentru afişarea numai a **divizorilor primi** ai unui număr *n*, algoritmul anterior se modifică prin eliminarea tuturor divizorilor *i* găsiţi la un moment dat, operaţia repetându-se până când se elimină toţi divizorii din numărul *n* (*n* să ajungă la valoarea 1). Paşii algoritmului sunt:

Pasul 1. *Se iniţializează şirul de numere, cu care se va împărţi* ***n****, cu primul divizor posibil, prin operaţia i ← 2.*

Pasul 2. *Dacă* ***i*** *îl divide pe* ***n****, atunci se afişează* ***i*** *şi, atât timp cât* ***n*** *se împarte la* ***i****, se împarte* ***n*** *la* ***i****, pentru a elimina toate puterile lui* ***i*** *din numărul* ***n****.*

Pasul 3. *Se trece la următorul divizor posibil, prin incrementarea lui* ***i*** *cu i ← i+1.*

Pasul 4. *Dacă n<>1, atunci se revine la* ***Pasul 4****; altfel, se termină algoritmul.*

De exemplu, dacă n=36, afişarea divizorilor proprii se desfăşoară astfel:

1. Se iniţializează împărţitorul cu 2, prin operaţia de atribuire i=2.
2. 2 îl divide pe 36 (36 mod 2 =0) ⇒ Afişează **2**.
3. 2 îl divide pe 36 ⇒ Se împarte *n* la 2: n=36/2=18.
4. 2 îl divide pe 18 ⇒ Se împarte *n* la 2: n=18/2=9.
5. 2 nu îl divide pe 9 ⇒ Se incrementează împărţitorul cu 1: i=2+1=3.
6. n<>1 (9<>1) ⇒ 3 îl divide pe 9 (9 mod 3 =0) ⇒ Afişează **3**.
7. 3 îl divide pe 9 ⇒ Se împarte *n* la 3: n=9/3=3.
8. 3 îl divide pe 3 ⇒ Se împarte *n* la 3: n=3/3=1.
9. 3 nu îl divide pe 1 ⇒ Se incrementează împărţitorul cu 1: i=3+1=4.
10. n=1 (1=1) ⇒ Se termină algoritmul.

```
întreg n,i;
început
 citeşte n;
 i ← 2;
 cât timp n<>1 execută
   dacă n mod i = 0
     atunci scrie i;
            cât timp n mod i = 0
                n ← n div i;
            sfârşit_cât_timp;
   sfârşit_dacă;
```

```
   i←i+1;
 sfârşit_cât_timp;
sfârşit.
```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru generarea divizorilor unui număr.

Enunțul problemei: *Să se descompună un număr **n**, introdus de la tastatură, în factori primi (se va afişa sub forma **a la puterea b**, unde **a**^{b} este unul dintre factorii primi).*

Pasul 1. *Se inițializează şirul de numere cu care se va împărți **n** cu primul divizor posibil, prin operația i ← 2.*

Pasul 2. *Dacă **i** îl divide pe **n**, atunci se afişează **i**, se inițializează contorul **k**, în care se numără puterea divizorului **i**, cu 0, prin operația k ← 0, şi atât timp cât **n** se împarte la **i** se incrementează contorul **k**, prin operația k ← k+1, şi se împarte **n** la **i**, pentru a elimina toate puterile lui **i** din numărul **n**, după care se afişează **k**.*

Pasul 3. *Dacă n<>1, atunci se revine la **Pasul 2**; altfel, se termină algoritmul.*

```
întreg n,i,k;
început
 citeşte n;
 i ← 2;
 cât timp n<>1 execută
   dacă n mod i = 0
     atunci k ← 0;
            cât timp n mod i = 0
               k ← k+1; n ← n div i;
            sfârşit_cât_timp;
            scrie i," la puterea ",k;
   sfârşit_dacă;
   i ← i+1;
 sfârşit_cât_timp;
sfârşit.
```



Probleme care se pot rezolva cu ajutorul algoritmului de prelucrare a divizorilor unui număr:

1. Să se scrie algoritmul prin care se afişează suma şi produsul divizorilor primi ai unui număr natural *n* care se introduce de la tastatură.
2. Să se scrie algoritmul prin care se determină toate numerele naturale perfecte mai mici decât un număr *n* introdus de la tastatură. Un număr natural se numeşte **număr perfect** dacă este egal cu suma divizorilor săi, din care se exclude divizorul egal cu numărul însuşi. Exemplu: 6=1+2+3; 28=1+2+4+7+14.
3. Se introduce de la tastatură un număr prim *p* şi se citesc pe rând de la tastatură mai multe numere naturale, până când se citeşte numărul 0. Să se determine numărul maxim *n*, astfel încât p^n să dividă produsul numerelor naturale introduse de la tastatură, fără să se calculeze produsul acestor numere.

4. Se citesc *n* numere naturale diferite de 0. Pentru fiecare număr citit să se afişeze divizorii pari. Dacă nu are divizori pari, să se afişeze un mesaj de informare.

3.3.7. Algoritmi pentru conversii între sisteme de numeraţie

Algoritmul pentru conversia din baza 10 în baza q

Conversia unui număr *n10*, din baza 10, într-un număr *nq*, reprezentat în baza *q* ($2 \le q \le 9$), se face prin împărţirea întreagă a numărului la baza q, până când restul obţinut este mai mic decât baza. Resturile obţinute în urma acestor operaţii de împărţire reprezintă cifrele reprezentării numărului în baza *q*, primul rest fiind cifra cea mai puţin semnificativă, iar ultimul rest, cifra cea mai semnificativă a reprezentării numărului. Pentru a scrie pe ecran numărul obţinut în urma conversiei, se va folosi scrierea lui ca un număr în baza 10 (cu cifrele corespunzătoare reprezentării în baza *q*). Paşii algoritmului sunt:

Pasul 1. *Se iniţializează numărul **nq** cu 0, prin operaţia nq ← 0.*
Pasul 2. *Se iniţializează **p**, puterea lui 10, cu 1, prin operaţia p ← 1.*
Pasul 3. *Se împarte **n10** la **q** şi se obţine restul **c**, care va fi una dintre cifrele reprezentării (c ← n mod q), şi câtul **n10** (n10 ← n10 div q).*
Pasul 4. *Se actualizează formatul de afişare a reprezentării numărului în baza **q**, prin operaţia nq ← nq+c*p.*
Pasul 5. *Se creşte puterea lui 10, prin operaţia p ← p*10.*
Pasul 6. *Dacă n10<>0, atunci se revine la **Pasul 3**; altfel, se afişează **nq**.*

De exemplu, dacă n10=11 şi q=2, conversia se desfăşoară astfel:
1. nq=0; p=1;
2. Se calculează c=11 mod 2=1 şi n10= 11 div 2=5.
3. Se actualizează formatul de afişare, nq= 0+1*1=1, şi puterea lui 10 p=1*10=10.
4. n10<>0 (5<>0) ⇒ Se calculează c=5 mod 2 = 1 şi n10=5 div 2=2.
5. Se actualizează formatul de afişare, nq=1+1*10=11, şi puterea lui 10, p=10*10=100.
6. n10<>0 (2<>0) ⇒ Se calculează c=2 mod 2=0 şi n10=2 div 2=1.
7. Se actualizează formatul de afişare, nq=11+0*100=11, şi puterea lui 10, p=100*10=1000.
8. n10<>0 (1<>0) ⇒ Se calculează c=1 mod 2=1 şi n10=1 div 2=0.
9. Se actualizează formatul de afişare, nq=11+1*1000=1011, şi puterea lui 10, p=1000*10=10000.
10. n10=0 (0=0) ⇒ Se afişează nq, adică 1011.

Algoritmul pentru conversia din baza q în baza 10

Conversia unui număr *nq*, din baza q ($2 \le q \le 9$), într-un număr *n10*, reprezentat în baza *10*, se face folosind descompunerea numărului după puterile bazei:

$$nq = a_n a_{n-1} \ldots a_1 a_0 = a_n \times q^n + a_{n-1} \times q^{n-1} + \ldots + a_1 \times q^1 + a_0 \times q^0$$

Numărul *n10* este o sumă în care termenii sunt produsele dintre cifra reprezentării în baza *q* şi puterea corespunzătoare a bazei. Deoarece citirea cifrelor se face începând cu cifra cea mai semnificativă, algoritmul foloseşte următorul mod de grupare a termenilor reprezentării numărului în baza 10:

$$n10 = (\ldots((a_n \times q + a_{n-1}) \times q + a_{n-2}) \times q + \ldots + a_1) \times q + a_0$$

Paşii care se execută sunt:

Pasul 1. *Se iniţializează numărul* ***n10*** *cu 0, prin operaţia n10 ← 0.*
Pasul 2. *Se citeşte cifra* ***c****.*
Pasul 3. *Se adună, la numărul* ***n10*** *înmulţit cu* ***q****, cifra citită, prin operaţia n10 ← n10*q+c.*
Pasul 4. *Se citeşte cifra* ***c****.*
Pasul 5. *Dacă* ***c*** *este o cifră (c>=0 şi c<q), atunci se revine la* ***Pasul 3****; altfel, se afişează numărul* ***n10****.*

De exemplu, dacă nq=1011 şi q=2, conversia se desfăşoară astfel:

1. n10=0;
2. Se citeşte c = 1
3. 0<=c<2 (0<=1<2) ⇒ Se calculează n10 = 0*2+1 = 1.
4. Se citeşte c = 0
5. 0<=c<2 (0<=0<2) ⇒ Se calculează n10 = 1*2+0 = 2.
6. Se citeşte c = 1
7. 0<=c<2 (0<=1<2) ⇒ Se calculează n10 = 2*2+1 = 5.
8. Se citeşte c = 1
9. 0<=c<2 (0<=1<2) ⇒ Se calculează n10 = 5*2+1 = 11.
10. Se citeşte c = 3
11. c<0 sau c>=2 (3>2) ⇒ Se afişează 11.

Algoritmii sunt:

n10 ⇒ nq

```
întreg n10,nq,p,q;
început
  citeşte n10;
  nq ← 0; p ← 1;
  cât timp n10<>0 execută
    nq ← nq + p*(n10 mod q);
    n10 ← n10 div q;
    p ← p*10;
  sfârşit_cât_timp;
  scrie nq;
sfârşit.
```

nq ⇒ n10

```
întreg c,n10,q;
început
  n10 ← 0;
  citeşte q;
  citeşte c;
  cât timp c>=0 and c<q execută
    n10 ← n10*q + c;
    citeşte c;
  sfârşit_cât_timp
  scrie n10;
sfârşit.
```

Probleme care se pot rezolva cu ajutorul algoritmilor de conversie între sisteme de numeraţie

1. Se citeşte de la tastatură un număr natural *n*. Să se afişeze reprezentarea lui în baza *q*, q ∈ [2, 9]. *q* se introduce de la tastatură.
2. Se citesc de la tastatură un număr natural *n* şi un număr *q*, q ∈ [2, 9]. Să se verifice dacă *n* poate fi considerat ca o reprezentare a unui număr în baza *q*.
3. Se citeşte de la tastatură un număr natural *n*. Să se afişeze câte cifre are reprezentarea lui în baza *q*, q ∈ [2, 9]. *q* se introduce de la tastatură.
4. Se citesc de la tastatură *q* (baza de numeraţie, q ∈ [2, 9]) şi mai multe numere naturale care reprezintă cifrele unui număr în baza *q*, până când numărul introdus nu mai poate fi considerat cifră în această bază de numeraţie. Să se afişeze numărul, reprezentat în baza 10.

5. Se citeşte de la tastatură un număr *n*, care este reprezentarea numărului în baza *q*, q ∈[2, 9]. Să se afişeze numărul, reprezentat în baza 10. *q* se introduce de la tastatură.
6. Se citeşte de la tastatură un număr *n*, care este reprezentarea numărului în baza *q*, q ∈[2, 9]. Să se afişeze câte cifre are reprezentarea lui în baza 10. *q* se introduce de la tastatură.
7. Se citeşte de la tastatură un număr *n*, care este reprezentarea numărului în baza *q*, q ∈[2, 9]. Să se afişeze numărul, reprezentat în baza p, p ∈[2, 9]. *q* şi *p* se introduc de la tastatură.

3.3.8. Algoritmi pentru generarea şirurilor recurente

Algoritmul pentru generarea termenilor **şirului lui Fibonacci** este un exemplu clasic de algoritm pentru **definirea recurentă** a termenilor unui şir. Şirul lui Fibonacci este format din termeni definiţi prin recurenţă, astfel:

$a_1=1$
$a_2=1$
$a_3= a_1+ a_2$
....
$a_n= a_{n-2}+ a_{n-1}$

Pentru generarea primilor *n* termeni ai şirului lui Fibonacci (n≥3), se vor genera repetat termenii de rang *i* ai şirului, cu 3≤i≤n. Se vor folosi trei variabile de memorie: a1 pentru a_{i-2}, a2 pentru a_{i-1} şi a3 pentru termenul curent a_i. După fiecare generare a unui termen se vor actualiza valorile din variabilele a1 şi a2. Paşii algoritmului sunt:

Pasul 1. *Se iniţializează termenii **a1** şi **a2**, prin atribuirile a1 ← 1 şi a2 ← 1.*
Pasul 2. *Se afişează termenii **a1** şi **a2**.*
Pasul 3. *Se iniţializează contorul **i** care numără termenii generaţi, prin operaţia i ← 3 (urmează să se calculeze termenul 3).*
Pasul 4. *Se calculează **a3**, prin operaţia de atribuire a3 ← a1+a2.*
Pasul 5. *Se afişează termenul **a3**.*
Pasul 6. *Se actualizează valorile pentru **a1** şi **a2**, prin atribuirile a1 ← a2 şi a2 ← a3.*
Pasul 7. *Se incrementează contorul cu 1, prin operaţia i ← i+1.*
Pasul 8. *Dacă i<=n, atunci se revine la **Pasul 4**; altfel, se termină algoritmul.*

De exemplu, dacă n=5, generarea termenilor se desfăşoară astfel:

1. a1=1; a2=1; Afişează **1**, **1**. Se iniţializează contorul i=3.
2. a3=1+1=2. Afişează **2**.
3. Se actualizează termenii a1 şi a2: a1=a2=1 şi a2=a3=2.
4. Se incrementează contorul cu 1: i=3+1=4.
5. i<=5 (4<=5) ⇒ Se calculează a3=1+2=3. Se afişează **3**.
6. Se actualizează termenii a1 şi a2: a1=a2=2 şi a2=a3=3.
7. Se incrementează contorul cu 1: i=4+1=5.
8. i<=5 (5<=5) ⇒ Se calculează a3=2+3=5. Se afişează **5**.
9. Se actualizează termenii a1 şi a2: a1=a2=3 şi a2=a3=5.
10. Se incrementează contorul cu 1: i=5+1=6.
11. i>5 (6>5) ⇒ Se termină algoritmul.

```
întreg a1,a2,a3,n,i;
început
 citeşte n;
 a1 ← 1; a2 ← 1;
 scrie a1,a2
 pentru i←3,n execută
    a3 ← a1+a2; scrie a3;
    a1 ← a2; a2 ← a3;
 sfârşit pentru;
sfârşit.
```

Probleme care se pot rezolva cu ajutorul algoritmilor de generare a şirurilor recurente

1. Să se afişeze toţi termenii şirului lui Fibonacci mai mici decât un număr natural *n* introdus de la tastatură.
2. Să se determine dacă un număr *n*, introdus de la tastatură, poate fi un termen al şirului lui Fibonacci.
3. Să se verifice dacă două numere naturale *n* şi *m* introduse de la tastatură (m>=n) pot fi termeni consecutivi ai şirului lui Fibonacci, fără a se calcula termenii şirului. (**Indicaţie**. Se execută operaţia inversă, de determinare a termenilor precedenţi: iniţializarea cu a3←m, a2←n, a1←m-n şi generarea cu a3←a2, a2←a1 şi a1←a3-a2 cât timp a1>0; dacă a3=a2=1, *m* şi *n* sunt termeni consecutivi ai şirului lui Fibonacci).
4. Se citesc trei numere întregi *a*, *b* şi *c*, care reprezintă coeficienţii unei ecuaţii de gradul 2, şi un număr natural *n*. Să se calculeze $S_n=x_1^n+x_2^n$, unde x_1 şi x_2 sunt rădăcinile ecuaţiei. Suma se calculează fără a se rezolva ecuaţia de gradul 2. Notăm cu S suma rădăcinilor (S=-b/a) şi cu P produsul rădăcinilor (P=c/a). Atunci: $S_n = S \times (x_1^{n-1}+x_2^{n-1}) - P \times (x_1^{n-2}+x_2^{n-2}) = S \times S_{n-1} - P \times S_{n-2}$. Ştiind că $S_0 = 1+1=2$ şi $S_1 = S$, se va folosi definiţia recurentă:

   $S_0=2$
   $S_1=S$
   $S_2=S \times S_1 - P \times S_0$
   ....
   $S_n=S \times S_{n-1} - P \times S_{n-2}$
5. Să se afişeze primii *n* termeni ai şirului (*n* se introduce de la tastatură): 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, ...
6. Să se afişeze primii *n* termeni ai şirului (*n* se introduce de la tastatură): 1, 2, 1, 1, 2, 3, 4, 3, 2, 1, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, ...

3.4. Eficienţa algoritmilor

Aţi văzut că pentru rezolvarea unei probleme se pot folosi mai mulţi algoritmi. În acest caz, se va alege **algoritmul cel mai eficient**.

Algoritmul cel mai eficient este acela care foloseşte cel mai puţin resursele calculatorului, şi anume:

✓ **Memoria internă**. În memoria internă se alocă spaţiu atât pentru datele folosite de algoritm, cât şi pentru codul executabil al programului (instrucţiunile în cod maşină).

✓ **Procesorul.** Timpul de utilizare a procesorului depinde de timpul necesar pentru executarea algoritmului.

Memorie internă

Din punct de vedere al gândirii algoritmului, pentru a face economie de această resursă, trebuie avute în vedere următoarele:

✓ Alegerea corectă a tipului de dată pentru fiecare variabilă de memorie folosită în algoritm.

✓ Rezolvarea problemei folosind cât mai puține variabile de memorie.

Ați văzut că, atunci când i se atribuie unei date un tip de dată, data capătă mai multe atribute, care determină **domeniul de definiție intern** al datei (mulțimea în care poate lua valori data). Tipul de dată ales influențează calitatea programului deoarece el determină dimensiunea zonei de memorie alocată, algoritmul de codificare și operatorii admiși pentru prelucrare. Din această cauză, la alegerea tipului de dată trebuie să se facă în două moduri **analiza datei**:

✓ **Logic** (la nivelul conceptual). Analiza se face pornind de la enunțul problemei și constă în identificarea **domeniului de definiție extern** al datei. De exemplu, în enunțul problemei se precizează că trebuie prelucrat un număr întreg pozitiv, cu valori cuprinse între 0 și 200. Acesta este domeniul de definiție extern al datei.

✓ **Fizic** (la nivelul reprezentării ei în memoria internă). Analiza se face pornind de la tipurile de date implementate în limbajul de programare, fiecare tip de dată având un **domeniu de definiție intern** al datei. Să presupunem că în limbajul de pro-gramare sunt implementate următoarele trei tipuri de date întregi: *tipul 1*, cu domeniul de definiție [-128, 127], reprezentat pe un octet; *tipul 2*, cu domeniul de definiție [0, 255], reprezentat pe un octet; și *tipul 3*, cu domeniul de definiție [-32768, 32767], reprezentat pe doi octeți. În urma analizei fizice, trebuie ales tipul de dată adecvat. **Regula** este: **se alege tipul de dată care consumă cea mai puțină memorie, astfel încât domeniul de definiție extern al datei să fie inclus în domeniul de definiție intern al datei.** Pentru exemplul prezentat, se va alege *tipul 2* de dată, deoarece numai *tipul 2* și *tipul 3* respectă condiția de incluziune a domeniilor de definiție ($[0, 200] \subset [0, 255]$ și $[0, 200] \subset [-32768, 32767]$, dar $[0, 200] \not\subset [-128, 127]$), iar *tipul 2* ocupă mai puțin spațiu de memorie (1 octet) decât *tipul 3* (2 octeți).

Problema pare neimportantă atunci când algoritmul folosește câteva variabile de memorie. Dar trebuie să vă gândiți că pentru rezolvarea problemelor complexe se pot folosi structuri de date în care se memorează foarte multe date elementare (de la câteva zeci până la milioane de date elementare). În acest caz este important dacă tipul de dată elementară este ales corect. Fiecare octet de care nu are nevoie o dată elementară poate să însemne un consum inutil de millioane de octeți pentru structura de date.

Procesorul

Este evident că timpul de execuție al unui algoritm depinde de câte valori ale datelor de intrare vor fi prelucrate. Într-o formulare de genul *„se prelucrează n valori numerice citite de la tastatură"* sau într-o formulare de genul *„se prelucrează mai multe numere întregi citite de la tastatură, până la întâlnirea valorii 0"*, la o execuție a algoritmului se pot prelucra numai 2 valori pentru data de intrare (dacă *n* are

valoarea 2, sau dacă se citesc numai două numere întregi diferite de 0), sau se pot prelucra 100 de valori pentru data de intrare (dacă *n* are valoarea 100, sau dacă se citesc 100 numere întregi diferite de 0). Se defineşte **dimensiunea datelor de intrare** ca fiind numărul de valori pentru datele de intrare ale unui algoritm. Pentru compararea timpului de execuţie a doi algoritmi care rezolvă aceeaşi problemă, se va folosi aceeaşi dimensiune a datelor de intrare.

În funcţie de complexitatea algoritmului, evaluarea timpului de execuţie se poate face prin:

a) numărul de operaţii elementare ale algoritmului, sau
b) timpul mediu al algoritmului.

O **operaţie elementară** este o operaţie sau o succesiune de operaţii care nu depind de caracteristicile problemei. De exemplu, poate fi operaţie elementară o operaţie aritmetică (adunare, scădere etc.), o operaţie de comparaţie, o operaţie de atribuire, sau un grup bine precizat de astfel de operaţii (5 operaţii de adunare, 2 operaţii de comparare, 10 operaţii de atribuire etc.). În schimb, *n* operaţii de atribuire nu reprezintă o operaţie elementară, deoarece depind de o caracteristică a problemei, şi anume de valoarea lui *n*.

Există cazuri în care nu se poate preciza numărul de operaţii elementare, acestea depinzând de valoarea datelor de intrare. Să comparăm următorii doi algoritmi:

Enunţul problemei 1. *Se citesc **n** numere. Să se calculeze suma lor.*
Enunţul problemei 2. *Se citesc **n** numere. Să se calculeze suma numerelor pare.*

Problema 1

```
întreg n,a,i,s;
început
 citeşte n;
 s ← 0;
 pentru i←1,n execută
   citeşte a;
   s ← s+a;
 sfârşit_pentru;
 scrie s;
sfârşit.
```

Problema 2

```
întreg n,a,i,s;
început
 citeşte n;
 s ← 0;
 pentru i←1,n execută
   citeşte a;
   dacă a mod 2=0
     atunci s ← s+a;
   sfârşit_dacă;
 sfârşit_pentru;
 scrie s;
sfârşit.
```

În cazul ambilor algoritmi se execută, în afara structurii repetitive, patru operaţii elementare (o operaţie de citire, o atribuire, o iniţializare contor şi o operaţie de scriere).

În cazul primului algoritm se execută în structura repetitivă patru operaţii elementare (două operaţii asupra contorului *i* – o comparare şi o incremetare –, o operaţie de citire şi o atribuire). În total, se execută 4+4xn operaţii.

În cazul celui de al doilea algoritm se execută în structura repetitivă două operaţii asupra contorului *i* – o comparare şi o incremetare –, o operaţie de citire, o operaţie de comparare, şi nu se poate preciza dacă se execută şi atribuirea). În total se execută 4+4xn+x operaţii, unde x depinde de câte numere pare se citesc de la tastatură.

În cel de al doilea caz pentru compararea algoritmilor se foloseşte **timpul mediu de execuţie** al algoritmului. Există un **timp minim de execuţie** care corespunde cazului cel mai favorabil (cazul în care se execută cele mai puţine operaţii) şi un **timp maxim de execuţie** care corespunde cazului cel mai defavorabil (cazul în care se execută cele mai multe operaţii). Pentru exemplul prezentat cazul cel mai favorabil este ca să nu existe nici un număr par. În acest caz x=0 şi numărul de operaţii elementare este 4+4×n. Cazul cel mai defavorabil este ca toate numerele să fie pare. În acest caz x=n şi numărul de operaţii elementare este 4+5×n. Pentru a calcula timpul mediu de execuţie, va trebui să analizăm toate cazurile posibile (în total *n+1* cazuri): nici un număr par, un număr par, două numere pare, ..., *n* numere pare, numărul mediu de operaţii fiind dat de *x* calculat astfel:

x = (0+1+2+....+n)/(n+1) = (n*(n+1)/2)/(n+1) =n/2

Studiu de caz

Scop: exemplificarea modului în care pot fi comparaţi doi algoritmi folosind numărul de operaţii elementare.

Enunţul problemei 1: *Să se determine valoarea unui polinom de gradul **n**. Gradul polinomului **n**, coeficienţii a_i ai polinomului şi valoarea lui **x** se introduc de la tastatură.*

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare**: *n* pentru gradul polinomului, *x* pentru valoarea pentru care se efectuează calculul şi *a* pentru citirea unui coeficient.
- ✓ **Data intermediară** *i*, de tip contor, pentru numărarea celor *n+1* coeficienţi citiţi.
- ✓ **Data de ieşire** *p* pentru valoarea polinomului.

Pentru rezolvarea acestei probleme se vor folosi două variante de algoritmi;

Varianta 1. $P(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1x^1 + a_0x^0$
Introducerea coeficienţilor începe cu coeficientul a_0 şi se termină cu coeficientul a_n.

Varianta 2. $P(x) = (...((a_n x + a_{n-1}) x + a_{n-2}) x + ... + a_1) x + a_0$
Introducerea coeficienţilor începe cu coeficientul a_n şi se termină cu coeficientul a_0.

Algoritmii sunt:

Varianta 1

```
întreg n,p,x,a,i;
început
 citeşte n,x,a;
 p ← a;
 pentru i←1,n execută
   citeşte a;
   p ← p+a*x;
   x ← x*x;
 sfârşit_pentru;
 scrie p;
sfârşit.
```

Varianta 2

```
întreg n,p,x,a,i;
început
 citeşte n,x,a;
 p ← a;
 pentru i←1,n execută
   citeşte a;
   p ← p*x+a;
 sfârşit_pentru;
 scrie p;
sfârşit.
```

Se observă că cei doi algoritmi diferă din punct de vedere al structurii repetitive, în care se execută, în prima variantă, un grup de cinci operaţii (două operaţii asupra

contorului *i* – o comparare şi o incremetare –, o citire şi două atribuiri), iar în a doua variantă, un grup de patru operaţii (două operaţii asupra contorului *i* – o comparare şi o incremetare –, o citire şi o atribuire), în afara structurii repetitive executându-se în ambele cazuri patru operaţii elementare (o operaţie de citire a trei date de intrare, o operaţie de atribuire, o operaţie de iniţializare a contorului şi o operaţie de scriere). Aşadar, pentru aceeaşi dimensiune a datelor de intrare (*n*), în prima variantă se execută 4+5×n operaţii elementare, iar în a doua variantă 4+4×n operaţii elementare. Este evident că a doua variantă este cea eficientă.

Enunţul problemei 2: *Se introduc **n** numere de la tastatură. Să se numere câte sunt divizibile cu 2, câte sunt divizibile cu 3 şi câte sunt divizibile cu 6.*

Se vor folosi următoarele variabile de memorie;

✓ **Date de intrare:** *n* pentru numărul de elemente şi *a* pentru citirea unui număr.
✓ **Data intermediară** *i*, de tip contor pentru numărarea celor *n* numere care se citesc.
✓ **Date de ieşire:** *k1* pentru numărarea numerelor divizibile cu 2, *k2* pentru numărarea numerelor divizibile cu 3 şi *k3* pentru numărarea numerelor divizibile cu 6.

Pentru rezolvarea acestei probleme se vor folosi două variante de algoritmi:

Varianta 1

```
întreg n,a,k1,k2,k3,i;
început
 citeşte n,a;
 k1 ← 0; k2 ← 0; k3 ← 0;
 pentru i=1,n execută
   citeşte a;
   dacă a mod 2 = 0
     atunci
       k1 ← k1+1;
       dacă a mod 3 = 0
         atunci
           k2 ← k2+1;
           k3 ← k3+1;
       sfârşit_dacă;
     altfel
       dacă a mod 3 = 0
         atunci
           k2 ← k2+1;
       sfârşit_dacă;
    sfârşit_dacă;
 sfârşit_pentru;
 scrie k1,k2,k3;
sfârşit.
```

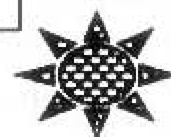
Varianta 2

```
întreg n,a,k1,k2,k3,i;
început
 citeşte n,a;
 k1 ← 0; k2 ← 0; k3 ← 0;
 pentru i=1,n execută
   citeşte a;
   dacă a mod 2 = 0
     atunci
       k1 ← k1+1;
   sfârşit_dacă;
   dacă a mod 3 = 0
      atunci
       k2 ← k2+1;
   sfârşit_dacă;
   dacă a mod 6 = 0
      atunci
       k3 ← k3+1;
      sfârşit_dacă;
   sfârşit_dacă;
 sfârşit_pentru;
 scrie k1,k2,k3;
sfârşit.
```

În acest caz, nu se poate preciza numărul de operaţii elementare ale fiecărui algoritm, deoarece el depinde de fiecare număr care se citeşte: dacă nu este divizibil nici cu 2 şi nici cu 3, dacă este divizibil numai cu 2, dacă este divizibil numai cu 3 sau dacă este divizibil cu 6. Pentru compararea celor doi algoritmi, vom calcula numărul de operaţii pentru fiecare caz de divizibilitate:

	Varianta 1				Varianta 2			
divizibil	nu	cu 2	cu 3	cu 6	nu	cu 2	cu 3	cu 6
comparații	2	2	2	2	3	3	3	3
atribuiri	0	1	1	3	0	1	1	3
total	2	3	3	5	3	4	4	6

Se observă că prima variantă este mai eficientă decât cea de a doua.



Studiu de caz

Scop: exemplificarea modului în care pot fi comparați doi algoritmi folosind timpii de execuție. Vom considera primele două variante de algoritmi pentru testarea unui număr prim.

În cele două variante se execută sigur:

✓ **Varianta 1**: 5 operații elementare: o operație de citire, două atribuiri, o comparație și o operație de scriere.

✓ **Varianta 2**: 5 operații elementare: o operație de citire, o atribuire, două comparații și o operație de scriere.

Cazul cel mai favorabil este ca numărul să fie par. În acest caz, se mai execută:

✓ **Varianta 1**: 3 operații elementare: două operații de comparație (una în structura repetitivă și una în structura alternativă) și o operație de atribuire – în total 8 operații.

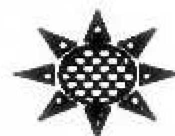
✓ **Varianta 2**: o operație de atribuire – în total 6 operații.

Cazul cel mai defavorabil este ca numărul să fie prim. În acest caz se execută:

✓ **Varianta 1**: de [sqrt(n)]-1 ori trei operații (două comparații și o atribuire), în total 3×([sqrt(n)]-1) operații– în total 5+3×([sqrt(n)]-1) operații.

✓ **Varianta 2**: de ([sqrt(n)]-2)/2 ori trei operații (două comparații și o atribuire), în total 3×(([sqrt(n)]-2)/2) operații– în total 5+3×(([sqrt(n)]-2)/2) operații.

Este evident că varianta 2 este mai eficientă.



Evaluare

Răspundeți:

1. De ce se folosesc cuvinte cheie la descrierea algoritmului prin pseudocod?
2. Dați câte un exemplu de problemă pentru fiecare dintre cele trei tipuri de structuri de control.
3. Verificați algoritmul lui Euclid de la pag. 56, pentru a=2 și b=162. Ce constatați? Modificați algoritmul astfel încât să fie mai eficient.

Adevărat sau Fals:

1. Pentru calculul modulului unui număr **x** folosiți o structură repetitivă.
2. Pentru calculul produsului a 10 numere folosiți o structură alternativă.

3. Pentru a vedea dacă un element **k** aparține unei mulțimi infinite de elemente, se folosește un algoritm prin care se verifică fiecare element al mulțimii, dacă este identic cu elementul **k**.
4. Pentru a vedea dacă un element **j** aparține unei mulțimi infinite de elemente, se construiește un algoritm prin care se verifică dacă elementul **j** corespunde unui criteriu prin care sunt definite elementele mulțimii.

Alegeți:

1. Fraza „cumpără acțiuni dacă produsul intern brut a crescut și vinde, în caz contrar" poate fi descrisă cu o:
   a) structură liniară b) structură alternativă c) structură repetitivă
2. Fraza „citește mai multe numere până când întâlnești numărul 0" poate fi descrisă cu:
   a) o structură repetitivă cu număr cunoscut de pași
   b) o structură alternativă generalizată
   c) o structură repetitivă cu număr necunoscut de pași
3. La un magazin se acordă un bonus cumpărătorilor: dacă valoarea cumpărăturilor (**v**) este între 5.000.000 lei și 10.000.000 lei, bonusul este de 1% din valoare, dacă valoarea este între 10.000.000 lei și 20.000.000 lei, bonusul este de 2% din valoare, iar dacă valoarea este mai mare de 20.000.000 lei, bonusul este de 5% din valoare. Precizați care dintre pseudocoduri descrie corect algoritmul de acordare a bonusului[1]:

```
a)
dacă v>5000000
 atunci
   v←v-v/100;
sfârşit_dacă;
dacă v>10000000
 atunci
   v←v-2*v/100;
sfârşit_dacă;
dacă v>20000000
 atunci
   v←v-5*v/100;
sfârşit_dacă;
```

```
b)
dacă v>5000000
 atunci
   dacă v>10000000
    atunci
      dacă >20000000
       atunci
         v←v-5*v/100;
       altfel
         v←v-2*v/100;
      sfârşit_dacă;
    altfel
      v←v-v/100;
   sfârşit_dacă;
sfârşit_dacă;
```

```
c)
dacă v>5000000
 atunci
  dacă v>10000000
    atunci
     dacă v>20000000
       atunci
        v←v-5*v/100;
     sfârşit_dacă;
    altfel
      v←v-2*v/100;
  sfârşit_dacă;
 altfel
   v←v-v/100;
sfârşit_dacă;
```

4. În urma executării secvenței: *p ← 2;* ***pentru*** *i ← 3, 5* ***execută*** *p ← p*i;* ***sfârşit_pentru;*** – valoarea lui ***p*** este:
   a) 16 b) 24 c) 120
5. În urma executării secvenței: *s ← 10;* ***pentru*** *i ← 2, 6* ***pas*** *2* ***execută*** *s ← s+i;* ***sfârşit_pentru;*** – valoarea lui *s* este:
   a) 16 b) 22 c) 30

---

[1] **Sugestie**: Verificați fiecare pseudocod alegând corect un set complet de date de intrare pentru testarea algoritmului. Alegerea setului complet de date de intrare se face dând valori corespunzătoare variabilei *v* pentru fiecare dintre cele patru cazuri de acordare a bonusului.

6. În urma executării secvenţei: *p ← 2;* ***pentru*** *i ← 1, 3* ***execută*** *p ← p*p;* ***sfârşit_pentru;*** – valoarea lui *p* este:
a) 16 b) 64 c) 256

7. În urma executării secvenţei: *i ← 3;* ***cât timp*** *i<10* ***execută*** *i ← i+3;* ***scrie*** *i;* ***sfârşit_cât_timp;*** – se afişează numerele:
a) 3, 6, 9 b) 6, 9, 12 c) 6, 9

8. În urma executării secvenţei: *i ← 3;* ***repetă scrie*** *i; i ← i+3;* ***până când*** *i>10;* – se afişează numerele:
a) 3, 6, 9 b) 6, 9, 12 c) 6, 9

9. Pentru n=3 şi x=2, în urma executării secvenţei: ***pentru*** *i ← 1, n* ***execută*** *x ← x*x;* ***sfârşit_pentru;*** – se afişează numerele:
a) 16 b) 256 c) 8

10. Pentru n=3 şi x=8, în urma executării secvenţei: ***pentru*** *i ← 1, n* ***execută*** *x ← x+i*i;* ***sfârşit_pentru;*** – se afişează numerele:
a) 15 b) 38 c) 22

11. Fie n=2033. Ca în urma executării secvenţei: ***cât timp*** *n >= m* ***execută*** *n←n-m;* ***sfârşit_cât timp;*** – variabila *n* să aibă valoarea 0, un număr *m* cu două cifre trebuie să aibă valoarea:
a) 99 b) 97 c) 19

Rezolvaţi:

1. Se consideră următorul algoritm, reprezentat în pseudocod (*n* este un număr natural):

```
citeşte n; m ← 0;
cât timp n<>0 execută
   n ← n div 2; m ← m+1;
sfârşit_cât_timp;
scrie m;
```

a) Ce valoare se va afişa pentru n=21?
b) Ce valori se pot citi pentru *n*, astfel încât să se afişeze 2?
c) Scrieţi pseudocodul unei structuri repetitive condiţionate posterior echivalente.

2. Se consideră următorul algoritm reprezentat în pseudocod (*a* şi *b* sunt numere naturale):

```
citeşte a,b;
dacă a>b atunci x ← a; a ← b; b ← x;
sfârşit_dacă;
n ← 0;
pentru i ← a,b execută
   dacă i mod 2 = 0 atunci n ← n+1;
   sfârşit_dacă;
sfârşit_pentru;
scrie n;
```

a) Ce valoare se va afişa pentru a=20 şi b=29?
b) Precizaţi o valoare pentru *a* şi o valoare pentru *b*, astfel încât să se afişeze 0.

c) Scrieţi în pseudocod un algoritm echivalent, care să folosească structura repetitivă condiţionată anterior şi un algoritm echivalent care să nu folosească nici o structură repetitivă.

3. Se consideră următorul algoritm reprezentat în pseudocod (*a* şi *b* sunt numere naturale):

```
citeşte a,b;
dacă a>b atunci x ← a; a ← b; b ← x;
sfârşit_dacă;
n ← 0;
cât timp a<b execută
   a ← a+1; b ← b-1; n ← n+1;
sfârşit_cât_timp;
scrie n;
```

a) Ce valoare se va afişa pentru a=17 şi b=25?
b) Precizaţi o valoare pentru *a* şi o valoare pentru *b*, astfel încât să se afişeze 0.
c) Scrieţi în pseudocod un algoritm echivalent, care să nu folosească nici o structură repetitivă.

4. Aplicarea algoritmilor

Puteți să rezolvați multe dintre problemele de la disciplinele şcolare cu ajutorul calculatorului. Pentru aceasta, trebuie mai întâi să descrieți rezolvarea lor cu ajutorul algoritmilor.

4.1. Rezolvarea problemelor de matematică

Enunţul problemei 1: *Se citesc trei numere întregi **a**, **b** şi **c**, care reprezintă coeficienții unei ecuații de gradul 2. Să se rezolve ecuația.*

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare** *a*, *b* şi *c* pentru coeficienții ecuației.
- ✓ **Data intermediară** *d* pentru discriminant.
- ✓ **Date de ieşire** *x1* şi *x2* pentru rădăcinile reale ale ecuației.

Algoritmul este:

```
real a,b,c,x1,x2,d;
început
 citeşte a,b,c;
 dacă a=0
   atunci
     scrie "Ecuatie de gradul intai";
     dacă b=0
       atunci
         dacă c=0
           atunci scrie " cu o infinitate de solutii";
           altfel scrie " care nu are solutii";
         sfârşit_dacă;                          // pentru c=0
       altfel
         x1 ← -c/b; scrie " cu radacina", x1;
      sfârşit_dacă;                             // pentru b=0
   altfel
     d ← b*b-4*a*c;
     dacă d>0
       atunci
         x1 ← (-b+sqrt(d))/(2*a); x2 ← (-b-sqrt(d))/(2*a);
         scrie "Ecuatia are doua radacini reale diferite ";
         scrie "x1= ", x1; scrie "x2= ", x2;
       altfel
         dacă d=0
           atunci
               x1 ← -b/(2*a);
               scrie "Ecuatia are doua solutii reale identice";
               scrie "x1=x2= ",x1;
```

```
          altfel
             scrie "Ecuatia nu are solutii reale ";
        sfârşit_dacă;                               // pentru d=0
     sfârşit_dacă;                                  // pentru d>0
  sfârşit_dacă;                                     // pentru a=0
sfârşit.
```

Pentru testarea algoritmului, folosiţi un set de date de intrare format din coeficienţii ecuaţiei (a,b,c), iar o mulţime completă de seturi de date de intrare poate fi: {(0, 0, 0), (0, 0, 1), (0, 2, 5), (1, -5, 6), (1, -2, 1), (1, 1, 1)}. Testaţi algoritmul.

Enunţul problemei 2: *Să se rezolve un sistem de două ecuaţii liniare cu două necunoscute:*

$$a_1 \times x + b_1 \times y = c_1$$
$$a_2 \times x + b_2 \times y = c_2$$

Soluţiile sistemului de ecuaţii sunt:

$$x = d_x/d_s = (b_2 \times c_1 - b_1 \times c_2)/(a_1 \times b_2 - b_1 \times a_2)$$
$$y = d_y/d_s = (a_1 \times c_2 - a_2 \times c_1)/(a_1 \times b_2 - b_1 \times a_2)$$

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare** *a1*, *b1*, *c1*, *a2*, *b2* şi *c2* pentru coeficienţii sistemului de ecuaţii.
- ✓ **Data intermediară** *ds, dx şi dy* pentru d_s, d_x şi d_y.
- ✓ **Date de ieşire** *x* şi *y* pentru soluţiile ecuaţiei.

Algoritmul este:

```
întreg a1,b1,c1,a2,b2,c2,ds,dx,dy;
real x,y;
început
 citeşte a1,b1,c1,a2,b2,c2;
 ds ← a1*b2-b1*a2;  dx ← b2*c1-b1*c2;  dy ← a1*c2-a2*c1;
 dacă ds=0
   atunci dacă dx=0
             atunci scrie "Sistem nedeterminat";
             altfel scrie "Sistem incompatibil";
          sfârşit_dacă;
   altfel
     x ← dx/ds; y ← dy/ds;
     scrie "x= ",x;  scrie "y= ",y;
 sfârşit_dacă;
sfârşit.
```

Enunţul problemei 3: *Se consideră două segmente de dreaptă în plan, neparalele cu axa Oy, reprezentate prin coordonatele lor. Să se determine intersecţia celor două segmente, dacă există, iar în caz contrar, să se specifice cazurile de excepţie.*

Coordonatele extremităţilor segmentelor sunt (x_1, y_1) şi (x_2, y_2) pentru primul segment, şi (x_3, y_3) şi (x_4, y_4) pentru al doilea segment. Coordonatele punctului de intersecţie sunt (x_i, y_i). Panta şi ordonata la origine ale dreptelor sunt m_1 şi n_1, pentru prima dreaptă, şi m_2 şi n_2 pentru a doua dreaptă.

Se vor folosi următoarele variabile de memorie;

✓ **Date de intrare**: *x1*, *y1*, *x2*, *y2*, *x3*, *y3*, *x4*, *y4* pentru coordonatele extremităţilor segmentelor.

✓ **Date intermediare**: *m1* şi *m2* pentru pantele segmentelor, *n1* şi *n2* pentru ordonatele la origine ale segmentelor, *min1* şi *max1* pentru minimul şi maximul dintre abscisele primului segment, *min2* şi *max2* pentru minimul şi maximul dintre ordonatele primului segment, *min3* şi *max4* pentru minimul şi maximul dintre abscisele celui de al doilea segment, *min4* şi *max4* pentru minimul şi maximul dintre ordonatele celui de al doilea segment.

✓ **Date de ieşire**: *x* şi *y* pentru coordonatele punctului de intersecţie.

Pasul 1. *Se citesc coordonatele extremităţilor segmentelor.*

Pasul 2. *Dacă segmentele sunt paralele cu axa Oy (x1=x2* **or** *x3=x4) scrie mesajul "Cel puţin o dreaptă este paralelă cu Oy" şi termină algoritmul.*

Pasul 3. *Se calculează pantele* ***m1*** *şi* ***m2****.*

Pasul 4. *Dacă pantele sunt egale (**m1** = **m2**), treci la **Pasul 5**; altfel, treci la **Pasul 6**.*

Pasul 5. *Dacă, un punct de pe prima dreapă se găseşte pe a doua dreaptă (coordonatele lui verifică ecuaţia celei de a doua drepte, atunci scrie mesajul "Segmentele sunt pe aceeaşi dreaptă"; altfel, scrie mesajul "Segmentele sunt paralele". Termină algoritmul.*

Pasul 6. *Se calculează coordonatele punctului de intersecţie al dreptelor.*

Pasul 7. *Dacă abscisa punctului de intersecţie nu este între abscisele capetelor unuia dintre segmente, sau dacă ordonata punctului de intersecţie nu este între ordonatele capetelor unuia dintre segmente, atunci scrie mesajul "Segmentele se intersectează pe prelungiri".*

Pasul 7. *Scrie coordonatele punctului de intersecţie.*

Algoritmul este:

```
real x1,y1,x2,y2,x3,y3,x4,y4,x,y,m1,n1,m2,n2,min1,max1,
     min2,max2,min3,max3,min4,max4;
început
 citeşte x1,y1,x2,y2,x3,y3,x4,y4;
 dacă (x1=x2) or (x3=x4)
   atunci scrie "Cel puţin o dreaptă este paralelă cu Oy";
   altfel
      m1 ← (y2-y1)/(x2-x1); m2 ← (y4-y3)/(x4-x3);
      n1 ← y1-m1*x1; n2 ← y3-m2*x3;
      dacă m1=m2
        atunci
          dacă y1=m2*x1+n2
            atunci scrie "Segmentele sunt pe aceeaşi dreaptă";
            altfel scrie "Segmentele sunt paralele";
          sfârşit_dacă;
        altfel
          x ← (n2-n1)/(m1-m2); y ← m1*x+n1;
          dacă x1<x2
            atunci min1 ← x1; max1 ← x2;
            altfel min1 ← x2; max1 ← x1;
          sfârşit_dacă;
```

```
            dacă y1<y2
              atunci min2 ← y1; max2 ← y2;
              altfel min2 ← y2; max2 ← y1;
            sfârşit_dacă;
            dacă x3<x4
              atunci min3 ← x3; max3 ← x4;
              altfel min3 ← x4; max3 ← x3;
            sfârşit_dacă;
            dacă y3<y4
              atunci min4 ← y3; max4 ← y4;
              altfel min4 ← y4; max4 ← y3;
            sfârşit_dacă;
            dacă x<min1 or y<min2 or x<min3 or y<min4 or x>max1
                 or y>max2 or x>max3 or y>max4
              atunci scrie "Segmentele se intersecteaza pe
                                prelungiri";
            sfârşit_dacă;
            scrie "x= ",x; scrie "y= ",y;
        sfârşit_dacă;
    sfârşit_dacă;
sfârşit.
```

Alegeţi o mulţime completă de seturi de date de intrare şi testaţi algoritmul.

Probleme de matematică propuse:

1. Se citesc două numere, care reprezintă dimensiunea înălţimii şi a razei unui cilindru circular drept. Să se calculeze aria laterală, aria totală şi volumul cilindrului şi cele ale conului circular drept, de aceeaşi rază şi înălţime cu cilindrul.
2. Se citesc trei numere întregi *a*, *b* şi *c*. Dacă pot reprezenta laturile unui triunghi să se calculeze dimensiunile razei cercului înscris, a razei cercului circumscris, ale razelor cercurilor exînscrise şi ale înălţimilor.
3. Să se determine rădăcinile ecuaţiei $a*x^4+b*x^2+c=0$. Coeficienţii ecuaţiei se citesc de la tastatură.
4. Se citesc de la tastatură numărătorul *a* şi numitorul *b* ale unei fracţii. Să se afişeze fracţia simplificată.
5. Se citesc coordonatele unui punct din plan. Să se precizeze unde este situat punctul în plan (denumirea axei sau numărul cadranului).
6. Se citesc coordonatele a două puncte din plan. Să se calculeze distanţa dintre ele şi coordonatele mijlocului segmentului de dreaptă care le uneşte.
7. Se citesc coordonatele a trei puncte din plan. Să se precizeze dacă punctele sunt coliniare sau nu.
8. Se citesc coeficienţii ecuaţiei carteziene generale a unei drepte. Să se precizeze cum este dreapta: oarecare, prima bisectoare, a doua bisectoare, trece prin origine, paralelă cu Ox, paralelă cu Oy, axa Ox sau axa Oy.
9. Se citesc coordonatele vârfurilor unui triunghi. Să se determine coordonatele ortocentrului.

10. Se citesc coordonatele a două vârfuri ale unui triunghi şi coordonatele ortocentrului. Să se determine coordonatele celuilalt vârf.
11. Se citesc coeficienţii ecuaţiilor carteziene generale pentru trei drepte. Să se determine dacă dreptele sunt paralelele şi echidistante.
12. Se citesc coordonatele a trei puncte din plan. Să se precizeze dacă ele pot fi vârfurile unui triunghi. În caz afirmativ, să se spună de ce tip este triunghiul (oarecare, echilateral, isoscel, dreptunghic, dreptunghic isoscel) şi să se calculeze aria şi perimetrul triunghiului.

4.2. Rezolvarea problemelor de fizică

Enunţul problemei 1: *Două mobile pornesc din punctul A, se mişcă rectiliniu uniform în aceeaşi direcţie, cu vitezele* v_1 *şi* v_2 $(v_1 > v_2)$, *şi ajung simultan în punctul B, la distanţa* **d** *de A. Mobilul 1 pleacă mai târziu cu* t_0 *decât mobilul 2. Se consideră cunoscute:* v_1, v_2 *şi* **d**. *Să se calculeze* t_0. *Unitatea de măsură pentru viteze este km/h, iar pentru distanţă, km. Timpul se va exprima în ore, minute şi secunde.*

Se notează cu d distanţa dintre A şi B, cu t1 şi t2 timpul de mişcare pentru cele două mobile şi cu v1 şi v2 vitezele lor. Legea de mişcare pentru cele două mobile are forma:

(1) d=v1×t1
(2) d=v2×t2=v2× (t1+t0)

Rezultă că t0=d*(1/v1-1/v2). Unitatea de măsură pentru distanţă va fi *m*, pentru viteze *m/s*, iar pentru timp *s*.

Se vor folosi următoarele variabile de memorie;
- ✓ **Date de intrare**: *v1*, *v2* şi *d*.
- ✓ **Date de ieşire**: *th*, *tm* şi *ts* pentru reprezentarea timpului în ore, minute şi secunde.

Algoritmul este:

```
real v1,v2,d,th,tm,ts;
început
 citeşte v1,v2,d;
 v1 ← v1*5/18; v2 ← v2*5/18; d ← d*1000;
 th ← d*(1/v2-1/v1);
 dacă th>=3600
   atunci
     tm ← th-[(th/3600)]*3600;    // [x]- partea inteagă din x
     th ← [(th/3600)];
   altfel
      tm ← th; th ← 0;
 sfârşit_dacă;
 dacă tm>60
   atunci ts ← tm-[tm/60)]*60; tm ← [(tm/60)];
   altfel ts ← tm; tm ← 0;
 sfârşit_dacă;
 scrie "timp= ",th,"h ",tm,"m ",ts,"s";
sfârşit.
```

Enunțul problemei 2: *Două trenuri, având lungimile L_1 și L_2 și vitezele v_1 și v_2 ($v_2 > v_1$), se apropie unul de altul în mișcare rectilinie uniformă, pe linii paralele. Se consideră cunoscute: v_1, v_2 și $L_1=L_2=d$. Aflați timpul **t** scurs între momentul când se întâlnesc și momentul depășirii complete. Unitatea de măsură pentru viteze este km/h, iar pentru distanță m. Timpul se va exprima în secunde.*

Se notează cu *d* lungimea trenului, cu *t* timpul și cu *v1* și *v2* vitezele trenurilor. Unitatea de măsură pentru lungime va fi *m*, pentru viteze *m/s*, iar pentru timp *s*.

Există două variante:

Varianta 1. Trenurile se deplasează amândouă în aceeași direcție. Se notează cu *x* distanța parcursă de primul tren. Legea de mișcare pentru cele două trenuri are forma:

(1) x=v1×t
(2) x+2×d=v2×t

Rezultă că t=2×d/(v2-v1).

Varianta 2. Trenurile se deplasează în direcții opuse. Se notează cu *x1* distanța parcursă de primul tren și cu *x2* distanța parcursă de cel de al doilea tren. Legea de mișcare pentru cele două trenuri are forma:

(1) x1=v1×t
(2) x2 =v2×t

unde x1+x2=2×d. Rezultă că t=2×d/(v1+v2).

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare**: *v1*, *v2* de tip real, *d* de tip întreg și *opt* de tip caracter, pentru a stabili care dintre variante se alege (are valoarea "1" pentru prima variantă și "2" pentru a doua variantă).
- ✓ **Data de ieșire**: *t* de tip real.

Algoritmul este:

```
real v1,v2,t;
întreg d;
caracter opt;
început
 citeşte v1,v2,d,opt;
 în caz că opt
  caz "1": t ← 2*d/((v2-v1)*(5/18)); scrie t;
  caz "2": t ← 2*d/((v2+v1)*(5/18)); scrie t;
  altfel   scrie "Optiune eronata";
 sfârşit_în_caz_că;
sfârşit.
```

Probleme de fizică propuse:

1. Fie A și B două puncte pe dreaptă și ***d*** distanța dintre ele. Din A pleacă spre B un mobil cu viteza v_1. După timpul t_0 pleacă din B, în același sens un al doilea mobil, cu viteza v_2 ($v_2 > v_1$). Ambele mobile se deplasează în mișcare uniformă pe dreaptă. Se consideră cunoscute: v_1, v_2 și t_0. Se cere să se afle după cât timp ***t*** are loc întâlnirea vehiculelor, și distanța ***x*** de la A la punctul de întâlnire. Unitatea de măsură pentru viteze este km/h, pentru distanță km, iar pentru timp, h.

2. Fie A şi B două puncte pe dreaptă şi ***d*** distanţa dintre ele. Din A şi B pornesc în mişcare uniformă pe dreaptă, în acelaşi sens, două mobile, cu vitezele v_1 *şi* v_2 ($v_1 > v_2$), al doilea mobil, cu timpul t_0 mai târziu decât primul. Se consideră cunoscute: v_1, v_2 *şi* t_0. Se cere să se afle după cât timp ***t***, de la pornirea primului mobil, acesta îl ajunge pe al doilea, şi distanţele ***d1*** şi ***d2*** parcurse de mobile până la întâlnirea lor. Unitatea de măsură pentru viteze este m/s, pentru distanţă m, iar pentru timp s.
3. Fie A şi B două puncte pe dreaptă şi ***d*** distanţa dintre ele. Din A pleacă spre B un mobil cu viteza v_1 şi, după timpul t_0, din B pleacă spre A un al doilea mobil, cu viteza v_2. Ambele mobile se deplasează în mişcare uniformă pe dreaptă şi se întâlnesc la distanţa ***x*** de B. Se consideră cunoscute: ***d***, v_2 *şi* t_0. Se cere să se afle timpul ***t*** scurs de la plecarea primului mobil până la întâlnire, şi viteza v_1. Unitatea de măsură pentru viteze este km/h, pentru distanţă km, iar pentru timp s.
4. Două mobile pornesc simultan din punctul O în acelaşi sens, în mişcare uniformă, pe dreaptă, cu vitezele v_1 *şi* v_2 ($v_1 > v_2$). După timpul t_1, porneşte din O, în acelaşi sens, un al treilea mobil. El ajunge primul mobil cu timpul t_2 mai târziu decât pe al doilea. Se consideră cunoscute: v_1, v_2, t_1 *şi* t_2. Se cere să se afle viteza v_3 a maşinii a treia. Unitatea de măsură pentru viteze este km/h, iar pentru timp min.
5. Fie A şi B două puncte pe o dreaptă şi ***d*** distanţa dintre ele. Din A şi B pornesc simultan, unul spre altul, două mobile în mişcare uniformă. Ele se întâlnesc după timpul t_1 şi îşi continuă fiecare mişcarea. Mobilul plecat din A ajunge în punctul B cu timpul t_2 mai târziu decât ajunge în A mobilul plecat din B. Se consideră cunoscute: ***d***, t_1 *şi* t_2. Se cere să se afle vitezele v_1 şi v_2 ale mobilelor. Unitatea de măsură pentru viteze este m/s, pentru distanţă m, iar pentru timp min.
6. Fie A şi B două puncte pe o dreaptă şi ***d*** distanţa dintre ele. Din A şi B pornesc simultan, unul spre altul, două mobile în mişcare uniformă. Ele se întâlnesc la distanţa d_1 de B şi îşi continuă fiecare mişcarea, ajungând în B şi respectiv în A. Se întorc fără oprire şi se întâlnesc a doua oară, la distanţa d_2 de A, la timpul ***t*** după prima întâlnire. Se consideră cunoscute: d_1, d_2 *şi* ***t***. Aflaţi vitezele v_1 şi v_2 ale mobilelor şi distanţa ***d***. Unitatea de măsură este pentru viteze m/s, pentru distanţă m, iar pentru timp s.
7. Fie A şi B două puncte pe o dreaptă şi ***d*** distanţa dintre ele. Din A şi B pornesc simultan, unul spre altul, două mobile în mişcare uniformă. Ele se întâlnesc la distanţa d_1 de A şi îşi continuă fiecare mişcarea, ajungând în B şi respectiv în A, unde staţionează timpul ***t***, apoi se întorc cu aceeaşi viteză şi se întâlnesc a doua oară, la distanţa d_2 de B. Se consideră cunoscute: d_1, d_2 *şi* ***t***. Aflaţi distanţa ***d*** dintre cele două puncte. Unitatea de măsură pentru distanţă este km, iar pentru timp, h.

5. Implementarea algoritmilor

5.1. Caracteristicile limbajului de programare

Pentru a putea executa cu ajutorul calculatorului algoritmii descrişi în pseudocod, aceştia trebuie **implementaţi într-un limbaj de programare**, adică trebuie să-i reprezentaţi cu ajutorul instrucţiunilor unui limbaj de programare (în exemplele următoare, limbajul **Pascal** şi limbajul **C++**). Transcrierea algoritmului într-un limbaj de programare se face cu un program specializat numit **editor de texte**. În acest mod, se obţine un fişier care conţine un text şi care se numeşte **program sursă**.

Pentru a putea rezolva o problemă cu ajutorul calculatorului, trebuie însă să folosiţi un **program executabil** (un program care să fie încărcat în memoria internă a calculatorului şi care să poată fi executat de procesor). Aceasta înseamnă că nu este suficient să transcrieţi algoritmul din limbajul pseudocod în limbajul de programare, ci trebuie să mai executaţi şi alte operaţii:

- ✓ **Compilarea**, adică traducerea fiecărei instrucţiuni din limbajul de programare într-un grup de instrucţiuni în limbajul maşinii, obţinându-se **programul obiect**.
- ✓ **Editarea legăturilor**, adică asamblarea pieselor rezultate în urma compilării, pentru a se obţine programul executabil.

Aceste operaţii se realizează cu ajutorul a două programe, numite **compilator** şi **editor de legături**. Ansamblul de programe compilator şi editor de legături funcţionează ca un translator între două persoane care vorbesc limbi diferite. Dacă în cazul unui translator traducerea se face între două limbaje naturale, în cazul acestor programe **traducerea** se face între două **limbaje artificiale**: limbajul de programare şi limbajul maşinii. Compilatorul reprezintă dicţionarul pe baza căruia se realizează traducerea cuvintelor, iar editorul de legături asamblează aceste cuvinte în construcţii gramaticale corecte.

La fel ca şi un limbaj natural, orice limbaj de programare este caracterizat de:

- ✓ **Sintaxă.** Este formată din totalitatea regulilor de scriere corectă, astfel încât să se realizeze construcţii acceptate de compilator (construcţii pe care compilatorul să le poată traduce în cod maşină).
- ✓ **Semantică.** Reprezintă semnificaţia construcţiilor corecte din punct de vedere sintactic.
- ✓ **Vocabular.** Este format din totalitatea cuvintelor care pot fi folosite într-un program.

Setul de caractere folosit pentru construirea cuvintelor este format din:

- ✓ **literele mari şi mici ale alfabetului latin** (a÷z, A÷Z)
- ✓ **cifrele sistemului de numeraţie zecimal** (0÷9)
- ✓ **caracterele speciale** (spaţiu, +, -, *, /, %, =, !, #, ", (,). [,], ;, :, ., etc.)

Cuvintele pot fi:
- ✓ **identificatori**
- ✓ **separatori**
- ✓ **cuvinte cheie.**

Identificatorii sunt nume de obiecte folosite în program (cum sunt, de exemplu, variabilele de memorie). În cazul ambelor limbaje de programare, identificatorul este o succesiune de litere, cifre sau caracterul de subliniere (_), din care prima trebuie obligatoriu să nu fie cifră. De exemplu, sunt considerați identificatori corecți: a1, A1, delta, _nr, nr_prim, dar nu sunt considerați corecți identificatorii 1A, A B, A&B sau c.m.m.d.c..

Separatorii se folosesc pentru a delimita unitățile lexicale (o unitate lexicală este formată dintr-un șir de caractere care are o semnificație lingvistică). În ambele limbaje de programare se folosesc ca separatori:
- ✓ spațiul și caracterul tabulator (TAB), pentru a separa cuvintele;
- ✓ ; pentru a separa instrucțiunile (corespunde punctului care separă propozițiile, din limbajul natural);
- ✓ ansamblul de caractere de control CR+LF generat la apăsarea tastei **Enter** pentru a separa liniile de text (corespund scrierii unui paragraf de la începutul rândului, din limbajul natural).

Cuvintele cheie sunt identificatori cu o semnificație precisă pentru limbaj, care nu pot fi folosiți într-un alt context decât cel permis de semantica limbajului.

Limbajul Pascal	Limbajul C++
and, or, not, div, mod, var, array, string, file, record, type, const, begin, end, if, then, else, case, for, to, downto, while, do, repeat, until, break, exit, with etc.	**int, char, float, double, sizeof, void, typedef, struct, const, if, else, switch, case, default, for, while, do, break** etc.

Programul poate conține și comentarii. **Comentariile** sunt texte explicative folosite în program pentru a-l face mai ușor de înțeles. Ele nu sunt interpretate de compilator. Pot fi scrise oriunde în program și sunt delimitate de restul programului, astfel:

Limbajul Pascal

se folosesc perechile de caractere **{** și **}** sau **(*** și ***)**

```
{ comentariu }
(* comentariu *).
```

Limbajul C++

se folosesc perechile de caractere **/*** și ***/** pentru comentariul scris pe mai multe rânduri sau caracterele **//** pentru comentariul scris pe un rând

```
// comentariu
/* comentariu
comentariu */
```

5.2. Structura programului

Pentru a înțelege structura unui program, vom defini mai întâi **blocul**.

Blocul este unitatea de bază a oricărui program Pascal sau C++.

Blocul este format din două părți:

- ✓ **Partea declarativă.** Conține definiții de elemente necesare algoritmului pentru rezolvarea problemei: tipuri de date, constante, variabile de memorie etc. Definirea lor se face cu ajutorul **instrucțiunilor declarative.**
- ✓ **Partea executabilă** sau **partea procedurală.** Conține instrucțiunile care descriu pașii algoritmului care trebuie implementat pentru rezolvarea problemei. Aceste instrucțiuni se numesc **instrucțiuni imperative.** Ele sunt:
  a) **instrucțiunea de atribuire** prin care se atribuie unei variabile de memorie valoarea unei expresii (corespunde operației de atribuire din algoritm);
  b) **instrucțiunea expresie** prin care se evaluează o expresie (numai în limbajul C++);
  c) **instrucțiunea de control** prin care se modifică ordinea de execuție a programului (corespunde unei structuri de control dintr-un algoritm);
  d) **instrucțiunea procedurală** (prin care se cere execuția unui subprogram; subprogramul este un program care poate fi apelat dintr-un alt program, apelarea lui însemnând o cerere de executare).

Ambele limbaje de programare permit definirea unei instrucțiuni compuse. **Instrucțiunea compusă** este o instrucțiune formată dintr-un grup de instrucțiuni care sunt interpretate de compilator ca o singură instrucțiune. Definirea se face prin încadrarea grupului de instrucțiuni, de:

Limbajul Pascal	Limbajul C++
cuvintele cheie **begin ... end;**	parantezele **{ ... }**

Structura programului este:

Limbajul Pascal

Orice program este **un bloc.**

Blocul începe cu partea declarativă. Instrucțiunile pentru declararea variabilelor de memorie încep cu cuvântul cheie **var.**

Instrucțiunile imperative din partea executabilă sunt încapsulate într-o instrucțiune compusă delimitată de cuvintele cheie **begin ... end.**.

Programul poate să aibă un antet prin care i se atribuie un nume. Antetul programului are sintaxa:

program *nume;*

Limbajul C++

Orice program este o **colecție de definiții de variabile și funcții. Funcția** este un bloc precedat de un **antet. Blocul** este încapsulat într-o instrucțiune compusă. Așadar, programul conține entități de forma:

antet funcție
{ *bloc* }

În **antet** se precizează numele funcției, tipul rezultatului pe care-l întoarce prin chiar numele său și, eventual, parametrii de execuție (valori care se transmit blocului și care sunt necesare când se execută):

tip_rezultat nume (listă_parametri)

Una dintre funcții este **funcția rădăcină.** Ea este obligatorie și este primul bloc cu care începe execuția programului. Numele său este **main().** Antetul acestei funcții este:

void main()

ceea ce semnifică faptul că funcția nu întoarce nici un rezultat (**void**) și nu necesită parametri pentru apelare – parantezele () nu conțin lista de parametri.

Enunțul problemei: *Se citesc două numere întregi **a, b**. Să se calculeze suma lor.*

Programul în limbajul **Pascal** este:

`program ex 1;`	**Antetul programului** (nu este obligatoriu).
`var a,b,s:integer;`	**Partea declarativă**; conține o instrucțiune declarativă pentru variabilele de memorie *a*, *b* și *s* de tip întreg.
`begin`	**Începutul părții executabile.**
`write('a= ');` `readln(a);` `write('b= ');` `readln(b);` `s:=a+b;` `write('suma= ',s);`	Operațiile de **citire** și **scriere** se execută cu **instrucțiunile procedurale readln()** și **write()**. Aceste instrucțiuni apelează subprograme implementate în bibliotecile limbajului de programare. **Instrucțiune de atribuire.**
`end.`	**Sfârșitul părții executabile.**

Programul în limbajul **C++** este:

`void main()`	**Antetul funcției rădăcină** (este obligatoriu).
`{`	**Începutul blocului.**
`int a,b,s;`	**Partea declarativă**; conține o instrucțiune declarativă pentru variabilele de memorie *a*, *b* și *s* de tip întreg.
	Începutul părții executabile
`cout<<"a= ";` `cin>>a;` `cout<<"b= ";` `cin>>b;` `s=a+b;` `cout<<"suma= "<<s;`	Operațiile de **citire** și **scriere** se execută cu **instrucțiuni expresie**, care creează fluxuri de date implementate în bibliotecile limbajului de programare (**cin<<** și **cout>>**). Corespund unei **instrucțiuni procedurale** din alte limbaje. **Instrucțiune expresie** pentru operația de **atribuire**.
`}`	**Sfârșitul blocului.**

Observații. Parametrii cu care se apelează procedurile, respectiv se creează fluxurile de date pentru operațiile de scriere și citire, pot fi:

- ✓ pentru **citire**: numai nume de variabile de memorie (de exemplu, *a* și *b*).
- ✓ pentru **scriere**: nume de variabile de memorie (de exemplu, *s*) sau constante (de exemplu, constanta de tip șir de caractere 'a= ' în limbajul Pascal și "a= " în limbajul C++).

5.3. Structurile de control

Limbajele de programare au implementate instrucțiuni de control pentru reprezentarea structurilor de control alternative și repetitive.

Enunțul problemei: *Se citește un număr întreg **n**. Să se calculeze inversul său.* Acesta este un exemplu de implementare în limbajul de programare a **structurii alternative simple.**

Limbajul Pascal

```
var n:integer;
    inv:real;
begin
  write('n= '); readln(n);
  if n<>0 then inv:=1/n
          else inv:=0;
  write('invers= ',inv:5:3);
end.
```

Limbajul C++

```
void main()
{int n;
 float inv;
 cout<<"n= "; cin>>n;
 if (n!=0) inv=1./n;
      else inv=0;
 cout<<"invers= "<<inv;
}
```

Observații

1. În partea declarativă a fost declarată o variabilă de memorie de tip real (*inv*) și o variabilă de tip întreg (*n*).
2. Se pot scrie două instrucțiuni pe același rând (în exemplu, s-au scris pe același rând instrucțiunile de scriere și de citire).
3. În limbajul Pascal, pentru afișarea variabilei de memorie de tip real *inv*, s-a folosit în instrucțiunea pentru scriere cu format de afișare: *inv:5:3* – afișarea se face pe cinci poziții (una pentru partea întreagă, una pentru separatorul dintre partea întreagă și partea fracționară și 3 pentru partea fracționară). În limbajul C++ în instrucțiunea de atribuire *inv=1./n* s-a scris constanta de tip real (*1.*) pentru ca operatorul **/** să efectueze împărțirea reală și nu împărțirea întreagă.

Enunțul problemei: *Se citesc* ***n*** *numere întregi. Să se calculeze suma lor.* Acesta este un exemplu de implementare în limbajul de programare a **structurii repetitive cu număr cunoscut de pași.**

Limbajul Pascal

```
var n,s,i,a:integer;
begin
 write('n= '); readln(n);
 s:=0;
 for i:=1 to n do
  begin
   write('a= '); readln(a);
   s:=s+a;
  end;
 write('suma= ',s);
end.
```

Limbajul C++

```
void main()
{
 int n,i,s,a;
 cout<<"n= "; cin>>n;
 s=0;
 for (i=1;i<=n;i++)
  {cout<<"a= "; cin>>a;
   s=s+a;}
 cout<<"suma= "<<s;
}
```

Observație. În ambele limbaje, corpul instrucțiunii de control nu poate să conțină decât o singură instrucțiune. Dacă trebuie să se execute mai multe instrucțiuni, ele sunt încapsulate într-o instrucțiune compusă, delimitată de **begin ... end;** în limbajul Pascal, respectiv de **{ ... }**, în limbajul C++.

Enunțul problemei: *Se citesc mai multe numere întregi până se citește numărul 0. Să se calculeze suma lor.* Acesta este un exemplu de implementare în limbajul de programare a **structurii repetitive cu număr necunoscut de pași condiționată anterior**.

Limbajul Pascal

```
var s,a:integer;
begin
 s:=0;
 write('a= '); readln(a);
 while a<>0 do
  begin
   s:=s+a;
   write('a= '); readln(a);
  end;
 write('suma= ',s);
end.
```

Limbajul C++

```
void main()
{
 int s,a;
 s=0;
 cout<<"a= "; cin>>a;
 while (a!=0)
   {s=s+a;
    cout<<"a= "; cin>>a;}
 cout<<"suma= "<<s;
}
```

Cuprins

Referenţi: – prof. gr. I **Brânduşa Elena Mariana Bogdan** – inspector de specialitate, Inspectoratul Şcolar al Municipiului Bucureşti
– prof. gr. I **Emma Gabriela Dornescu** – Colegiul Tehnic *Petru Rareş*, Bucureşti; metodist de specialitate – Informatică – Inspectoratul Şcolar al Municipiului Bucureşti

Control ştiinţific: ing. **Liliana Dăbuleanu**

Descrierea CIP a Bibliotecii Naţionale a României
MILOŞESCU, MARIANA
Informatică: clasa a IX-a, Mariana Miloşescu. - Ed. a 3 - a
Bucureşti: Editura Didactică şi Pedagogică, 2006
ISBN (10) 973-30-1446-X; ISBN (13) 978-973-30-1446-1

004(075.35)
004.43(075.35)

EDITURA DIDACTICĂ ŞI PEDAGOGICĂ, R.A.
Str. Spiru Haret, nr. 12, sectorul 1, cod 010176, Bucureşti
Tel.: 021.315.38.20
Tel./Fax: 021.312.28.85; 021.315.73.98
E-mail: edp@b.astral.ro
www.edituradp.ro

Comenzile pentru această lucrare se primesc:
• prin poştă, pe adresa editurii
• prin e-mail: comenzi@edituradp.ro, edpcom@b.astral.ro
• prin telefon / fax: 021.315.73.98 (comercial)
• prin telefon: 021.315.38.20

Redactor: ***Liana Fâcă***
Coperta: ***Elena Drăgulelei-Dumitru***
Tehnoredactare computerizată: ***Mariana Miloşescu***

Acest manual este proprietatea Ministerului Educaţiei şi Cercetării. Manualul este aprobat prin Ordinul nr. 3886 din 24.05.2004, în urma licitaţiei organizate de către Ministerul Educaţiei şi Cercetării, este realizat în conformitate cu programa analitică aprobată de Ministerul Educaţiei şi Cercetării prin Ordinul nr. 3458 din 09.03.2004 şi este distribuit **gratuit** elevilor.

ACEST MANUAL A FOST FOLOSIT DE:						
Anul	Numele elevului care a primit manualul	Clasa	Şcoala	Anul şcolar	Starea manualului*	
					la primire	la returnare
1.	[illegible]	IX A	[illegible]	2013 2014	[illegible]	[illegible]
2.						
3.						
4.						

* Starea manualului se va înscrie folosind termenii: nou, bun, îngrijit, nesatisfăcător, deteriorat.

Profesorii vor controla dacă numele elevului este scris corect.

Elevii nu trebuie să facă nici un fel de însemnări pe manual.

ISBN 973-30-1446-X
ISBN 978-973-30-1446-1



1,56 RON